UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

DEPARTAMENTO DE TECNOLOGÍA ELECTRÓNICA

PROYECTO FIN DE CARRERA

MEDIDOR DE DISTANCIA POR ULTRASONIDOS

INGENIERÍA TÉCNICA INDUSTRIAL:
ELECTRÓNICA INDUSTRIAL

AUTOR: JAVIER SÁEZ CARDADOR
TUTOR: GUILLERMO CARPINTERO DEL BARRIO

JUNIO, 2006

AGRADECIMIENTOS

Quisiera comenzar agradeciendo a mi familia su esfuerzo por hacer una realidad de este sueño; apoyándome en los buenos y los malos momentos y ayudándome en todo lo que ha estado en sus manos en estos años.

Me gustaría agradecer la ayuda de mi tutor, Guillermo Carpintero, en la realización y materialización de este proyecto. Y junto a él, expresar mi agradecimiento a todos los profesores que durante estos años me han transmitido sus conocimientos y me han hecho crecer como persona.

También quisiera agradecer a toda la gente que se ha cruzado en mi camino a lo largo de estos años. A los que me han apoyado y hecho feliz y a los que no lo han hecho tanto. Ambos me han servido en todo momento para aprender y me han traído hasta aquí.

Y entre toda esa gente, me gustaría terminar haciendo especial mención a aquellos compañeros de fatigas como Víctor Plácer, Raúl Palencia, Leandro Boyano, Diana Plazuelo y Luís Giménez. Muchas gracias a todos por vuestra contribución a este proyecto, por vuestra amistad y por todos los momentos compartidos.

ÍNDICE

Pág.

ÍNDICE DE FIGURAS

Pág.

ÍNDICE DE TABLAS

Pág.

CAPÍTULO 1

INTRODUCCIÓN

1. INTRODUCCIÓN

Este proyecto se centra en la medición de distancias a través de ondas de ultrasonidos, usando el principio de funcionamiento del SONAR. Las características de este sistema son:

- Mediciones en rangos medios (Orden de centímetros).
- Velocidad de refresco de medida medio (Orden de milisegundos).
- Precisión de medida media (Orden de centímetros).

Si bien, este es un instrumento común de corte clásico; con este proyecto se pretenden utilizar nuevos recursos y enfoques de estos sistemas para adaptarlos a usos y aplicaciones más modernas y de actualidad.

Para conocer mejor las bases y antecedentes de este proyecto es conveniente remontarse en la historia y revisar las distintas definiciones de metro; así como los instrumentos desarrollados para la medida de distancias.

Algo tan simple como obtener la medida de esta memoria mediante una regla graduada, no ha sido siempre tan trivial. El hecho de medir es la comparación de un objeto en cuestión con una referencia.

Sin embargo, en la antigüedad, dichas referencias solían ser variables y tales como antebrazos, pies o palmos. Ya en Egipto o en la Edad Media se construyeron herramientas para poder comparar con objetos y obtener medidas parciales de distancias.

Pero no fue hasta el siglo XVIII cuando en plena revolución Francesa se detectó la necesidad de un sistema métrico estable y universal. Este sistema métrico debería cumplir los siguientes requisitos:

- Que estuviera basado en referencias que permanecieran estables e inmutables en la naturaleza.
- Que las distintas medidas de capacidad, peso, distancia, etc. se relacionasen entre sí fácilmente.
- Que siguiera un sistema decimal en el que las unidades cambiasen de 10 en 10.

Con estas premisas la Academia de Ciencias Francesa se dispuso a medir la longitud de un cuadrante polar terrestre, es decir, un cuarto de meridiano. De este modo se tomó la medición geodésica del meridiano que

atraviesa París entre Dunkerque y Barcelona y, a través de ella, se obtuvo la longitud total del cuarto de meridiano. Y finalmente, a la diez millonésima parte de dicha medida se la definió como metro en 1791 en París.

Con esta longitud llamada metro se consiguió basar la medida de distancias en una referencia invariable como es un meridiano terrestre. Y a través de el metro se establecieron más medidas como la del peso y la capacidad; definiéndose el kilogramo como el peso de un cubo de agua con lados de 10 centímetros y el litro como el volumen ocupado por dicho cubo.

Se impuso también un Sistema Métrico decimal a través de los prefijos *kilo, hecto, deca, deci, centi, mili,* etc. Así, a través de un estándar definido se podrían obtener distintas precisiones adecuadas a las magnitudes de los valores medidos.

Finalmente se construyeron una serie de patrones materiales como una barra o una pesa para establecer la referencia inmutable y estándar de las medidas y fueron cuidadosamente conservados.

No obstante, esto solo fue el principio del Sistema Métrico y la primera definición del metro. Fue sin duda el impulso inicial que permitió seguir avanzando técnicamente a través del tiempo, y así fijando constantes en mediciones. Así fue aceptado internacionalmente en la Convención del Metro celebrada en París en 1875.

De esta manera se llegó a que en 1960 el científico Alemán *Ernst Engelhard* consiguió, a través de las emisiones cuánticas visibles del isótopo *Criptón 86* y la interferometría; la transición de un patrón material único, como la barra de metal, a una definición cuántica muy estable y reproducible en cualquier lugar y situación con un mínimo error. Nació una nueva definición del metro que cumplía las mismas premisas.

Esta definición estuvo vigente hasta 1983 cuando, a través de comparaciones de emisiones cuánticas de distintos elementos, se estableció el valor de la velocidad de la luz en el vacío. Así se definió el metro como la distancia recorrida por la luz en el vacío en un intervalo de tiempo de 1/299.792.458 segundos con un error mínimo. Esta es la definición actual de metro.

Paralelamente a las diferentes definiciones del metro, la técnica avanzó en los distintos sistemas para la medición de distancias según fueron surgiendo las necesidades.

De tal modo, los primeros instrumentos de medida y los más usados para pequeñas distancias fueron las reglas calibradas, consistentes en una barra física marcada con las pertinentes subdivisiones para obtener la suficiente precisión. Así le siguieron las cintas métricas y demás instrumentos de medida directa y excesivamente simples.

Sin embargo, todos estos instrumentos no eran aptos para medir grandes distancias a gran velocidad o con buena precisión. En tiempos de guerra fueron estas las necesidades que se acrecentaron ya que, el hecho de conocer la posición del enemigo daba una gran ventaja.

Así empezaron a desarrollarse los primeros sistemas SONAR (*SOund Navigation And Ranging*) que se basaban en la emisión de ondas de ultrasonidos. En 1900 la empresa *Submarine Signal Company* desarrolló la primera aplicación comercial de un SONAR para conocer la distancia de alcance de barco a faro. En 1914 este dispositivo fue perfeccionado para que un barco pudiera detectar un iceberg a más de 3 kilómetros de distancia. Posteriormente sería evolucionado en la I Guerra Mundial.

Pronto se vieron ciertas limitaciones de los ultrasonidos en cuanto a distancias medidas y tiempos de respuesta. Es por ello que el SONAR dio el testigo en poco tiempo al RADAR (*RAdio Detection And Ranging*) con sus ondas de radio. En 1887 *Heinrich Hertz* ya estudiaba el principio del funcionamiento del RADAR.

Sin embargo, a partir de 1933 en la II Guerra Mundial el desarrollo del RADAR cobró la máxima importancia. Hechos destacados fueron el hito Ruso de detección de aviones hasta 70 kilómetros de distancia en 1934 o la exitosa demostración que el Inglés *Robert Watson Watt* hizo de este sistema en 1935. Muchos datan el primer RADAR en 1937 con la llamada *Chain Home*, es decir, la red de radares Inglesa contra aviones Alemanes.

Más tarde, con el desarrollo de los láseres, sobre todo los basados en semiconductor, las técnicas de medición fueron mejorando en rapidez, distancia y precisión. De los logros más importantes de estos sistemas se puede destacar el de la misión *Apollo XI* el 21 de Julio de 1969 a través del cual se midió por primera vez la distancia entre la Tierra y la Luna experimentalmente.

En la actualidad todos estos sistemas siguen conviviendo, cada uno en su nicho experimental y de mercado dependiendo de sus características y especificaciones.

1.1. MARCO TEÓRICO

Según lo mencionado, el proyecto que aquí se expone tiene como fin la medición de distancias por ultrasonidos. De tal modo, es imprescindible el conocimiento de dichas ondas para entender el funcionamiento de este sistema.

Una onda de ultrasonidos es una onda de presión cuya frecuencia se encuentra por encima del umbral de percepción para el oído humano (el límite se encuentra en torno a los 20 Khz.). Como onda de presión, independientemente de su frecuencia, una onda de ultrasonidos posee el mismo comportamiento que las ondas acústicas.

El ultrasonido, como onda de presión, posee las siguientes características:

- *Es una onda mecánica*: Esto significa que esta onda solo se propaga a través de un medio material y elástico que permita transmitir las vibraciones de sus partículas.

- *Es una onda longitudinal*: Esto quiere decir que el movimiento de las partículas que transporta la onda se desplaza en la misma dirección de propagación de la onda.

- *Es una onda esférica*: Esto hace referencia a que las ondas se desplazan en las tres dimensiones describiendo esferas radiales que parten del foco emisor.

La primera característica determina que la velocidad de propagación del sonido dependa del medio por que se transmita. Es decir, dependiendo de las propiedades del material por el que se propague la onda de sonido, su velocidad será variable.

Para el funcionamiento de este proyecto lo más interesante es estudiar esta velocidad en el aire. Según los estudios de dicha velocidad para una atmósfera estándar a una temperatura de 293,15 K la velocidad del sonido es de 344 m/s.

Una buena aproximación para calcular la velocidad del sonido en gases es la siguiente:

$$c = \sqrt{\frac{\kappa \cdot R \cdot T}{m}}$$

Donde,

- k es la relación de los calores específicos (para el aire k=1,4).
- R es la constante universal de los gases.
- T es la temperatura absoluta en Kelvin.
- m es el peso molecular promedio (para el aire R/m = 287 J/kgK).

Por otra parte, también es conveniente estudiar los fenómenos físicos que afectan a la propagación del sonido, bastante comunes en ondas. Estos fenómenos son los siguientes:

- *Reflexión*: Este fenómeno implica que, cuando una onda choca con una superficie lisa, esta es rebotada al medio del que proviene con un ángulo de reflexión igual al de incidencia.
- *Absorción*: Es el fenómeno por el cual una pequeña parte de la energía de la onda es absorbida al chocar con una superficie. El resto de la onda es reflejada.
- *Refracción*: Este fenómeno provoca el cambio de dirección de una onda de sonido al cambiar de medio de propagación. Este fenómeno se produce a causa del cambio de la velocidad de propagación que el sonido sufre al cambiar de medio.
- *Difracción*: Este fenómeno hace referencia a la dispersión que sufre una onda de sonido ante determinados obstáculos.

Todo esto sienta las bases físicas del presente proyecto. En este momento se está en disposición de describir las ondas de ultrasonidos con las que se va a trabajar así como las condiciones y premisas que se asumen a la hora de abordar el supuesto.

Primeramente conviene definir la onda de ultrasonidos con la que se trabajará. Esta definición se realiza en base a tres parámetros característicos, estos son los siguientes:

- *Tono*: Es la frecuencia del sonido. Será más agudo cuanto mayor sea la frecuencia y más grave cuanto menor sea la misma.
- *Intensidad*: Es la cantidad de energía de la onda. A mayor intensidad mayor será su volumen.

- *Duración*: Como su nombre indica, es el intervalo de tiempo durante el cual se prolonga el sonido.

La onda con la que se trabaja es una onda de frecuencia o tono de 40 KHz. Su duración será de 4 milisegundos. La intensidad depende de la excitación de la fuente del sonido y es la suficiente para el correcto funcionamiento de este proyecto.

En cuanto a la intensidad del sonido, habrá que tener en cuenta que disminuye de forma inversamente proporcional al cuadrado de la distancia que lo separa de la fuente. A esto se le llama *ley cuadrática inversa* aplicada al sonido.

Dado que el sonido se comporta como onda longitudinal, la emisión de una onda de ultrasonidos se esquematiza en la siguiente figura 1.1:

Figura 1.1: Esquema de emisión de una onda de ultrasonidos

En esta figura se observa como, tras emitir una onda de ultrasonidos en el instante T_0, esta viaja por el aire hasta rebotar con el primer obstáculo encontrado en el instante T_1. Por último, es rebotada en dicho obstáculo regresando hasta el punto donde se emitió en T_2.

A la onda recibida que ha rebotado contra el obstáculo se le llama eco y al tiempo total de propagación de la onda se le llama tiempo de eco, esto es T_2. Si se conoce este tiempo y se sabe la velocidad de propagación del sonido en el aire se podrá hallar la distancia recorrida por la onda. En este principio se basa el medidor diseñado en este proyecto.

Para el desarrollo del medidor se supondrá la velocidad del sonido constante e igual a 344 m/s independientemente del promedio de la masa del aire, la temperatura o la relación de calores específicos.

También se contará con el fenómeno de la reflexión sin el cual el efecto eco sería imposible y la medición de distancias a través de este sistema también.

No se pasarán por alto el efecto de absorción ni la ley cuadrática inversa teniendo en cuenta la atenuación de la intensidad de la onda.

Disponiendo de esta información queda introducido el marco teórico y la fundamentación física que hace posible el correcto funcionamiento del medidor de distancias y quedan sentadas las bases que permanecerán invariables durante el desarrollo de la memoria.

1.2. OBJETIVO DEL PROYECTO

Este proyecto tiene como objetivo el diseño y desarrollo de un medidor de distancias a través de ondas de ultrasonidos. Este desarrollo debe realizarse mediante dos dispositivos emisor y receptor de ultrasonidos. También se contará con un circuito digital sintetizado en una FPGA (*Field Programable Gate Array*) a partir del lenguaje de descripción VHDL (*VHSIC Hardware Description Languaje*).

Este sistema constará, por lo tanto, de dos bloques claramente diferenciados que son:

a) Emisor y receptor de ultrasonidos

b) Circuito y placa de la FPGA.

El primer bloque, bloque a, será el que interactúa con el entorno. Este se encargará de emitir la onda de ultrasonidos y a su vez, de recibirla al producirse el eco. Así mismo se deben acondicionar todas las señales eléctricas para que puedan ser conectadas al bloque b.

El segundo bloque, bloque b, es el que realiza el control del sistema. Gestionará las señales eléctricas que controlan al emisor y las que se producen en el receptor de ultrasonidos. De tal modo, dicho bloque debe ser capaz de dar la orden de emitir una serie de ondas de ultrasonidos y contar el tiempo que transcurre hasta que dichas ondas son recibidas por el receptor de ultrasonidos.

Posteriormente se tratará este tiempo para obtener el resultado deseado, es decir, la distancia al objeto al que se dirigen los ultrasonidos. Y por último, actuará como interprete entre el usuario y el medidor; de este modo adaptará el resultado final de la forma más adecuada para que pueda ser interpretada fácilmente por el usuario y actuará acorde con las órdenes recibidas de dicho usuario.

Por lo tanto, la medición por ultrasonidos desarrollada en este proyecto, es decir, el objetivo a conseguir; se centra en los siguientes hitos:

1. Emisión de varias ondas de ultrasonidos.
2. Recepción de las ondas emitidas y acondicionado como señal digital.
3. Cuenta del tiempo transcurrido entre emisión y recepción de las ondas.
4. Tratamiento del dato obtenido para conseguir hallar la distancia al objeto donde se reflejaron las ondas de ultrasonidos.
5. Tratamiento de la distancia para mostrar en un display de 4 dígitos de siete segmentos con punto la distancia medida en dos precisiones (Metros y Centímetros).
6. Interacción con el usuario a través del estado de varios interruptores.

La consecución de estos hitos, según lo descrito minuciosamente en el desarrollo de esta memoria, supone la resolución del medidor de distancias por ultrasonidos.

Los medios con los que se cuenta para la realización del proyecto son:

- Placa de desarrollo del fabricante de FPGAs *Xilinx*, manufacturada por la firma *Digilent* con software necesario.
- Emisor de ultrasonidos.
- Receptor de ultrasonidos.
- Todos aquellos circuitos y componentes electrónicos necesarios para el driver del emisor y el acondicionador del receptor de ultrasonidos.

1.3. ESPECIFICACIONES

Las especificaciones que debe cumplir el medidor de distancia son las que se muestran en la tabla 1.1, la cual se muestra a continuación:

Características del Medidor

Descripción	Valor	Unidades
Tensión de Alimentación	220	V AC
Potencia Consumida(1)	8	W
Tª de Funcionamiento(2)	0 - 70	ºC

Características de Medición

Descripción	Valor	Unidades
Rango de Medida	0,05 - 6	m.
Escala	cm. / m.	
Selector de Escala	Automático	
Error	Escala cm. / m.	cm. / m.
	±1 / ±0,1	
Mediciones por Segundo	≈3	Med. / s
Medición continua	Seleccionable	
Frecuencia de Ultrasonidos	40	KHz.

Notas:
1. Potencia máxima consumida por la fuente.
2. La temperatura óptima de funcionamiento es 25 ºC.

Tabla 1.1: Especificaciones del Medidor de Distancia

En esta tabla se muestran las especificaciones de funcionamiento del sistema. De este modo, junto con el epígrafe anterior, queda totalmente definido el producto final que se pretende desarrollar en este proyecto.

Conviene comentar de esta tabla que la selección automática de escala se produce al pasar la medida de los 99 cm. A partir de aquí el sistema comenzará a medir en metros. El cambio de escala se señalizará en el display de medición con un símbolo identificador para cada escala.

Por último, también hacer referencia a las mediciones por segundo. Este valor depende de la distancia a medir, ya que hay que contar con la variación del tiempo de recepción del eco en función de la distancia. Sin embargo, dada la velocidad del sonido, esta variación puede ser despreciable y aproximar estas mediciones por segundo a 3. Este es un valor que mantiene el compromiso para obtener mediciones estables pero con un buen refresco.

1.4. ORGANIZACIÓN DE LA MEMORIA

A continuación se comentará brevemente la estructura que se va a seguir en la realización de esta memoria.

Tras este Primer Capítulo de introducción, en el que se exponen los principios de funcionamiento y se definen los objetivos del proyecto; se detallarán el desarrollo electrónico y físico del sistema. Para ello se establece una clasificación de diseño: el hardware y la programación VHDL de la FPGA.

En el Capítulo Segundo se tratará el diseño hardware necesario para el funcionamiento del medidor. Así se describirá la función de todos sus bloques y componentes y como son utilizados para obtener el objetivo deseado. Por otra parte se ampliará todo este contenido con la implementación de dicho hardware físicamente en una placa de circuito impreso y con la descripción física y de hardware del equipamiento de la placa de desarrollo de la FPGA, la *Spartan III*.

En el Capítulo Tercero se especificará que es lo que el circuito de la FPGA tiene que cumplir exactamente para conseguir medir distancias como si de una caja negra se tratase. Se comentará estructuradamente toda la programación de la FPGA. Este capítulo incluye una descripción de las herramientas necesarias y procedimientos para programar la FPGA, explicando las bases del lenguaje usado. También

En el Capítulo Cuarto se evaluará el proyecto concretando las conclusiones y objetivos conseguidos a su finalización. Se comentarán las aplicaciones de este proyecto comercialmente o experimentalmente como parte de otros proyectos integrándose en ellos. Posteriormente se describirán posibles trabajos futuros y ampliaciones de cara a mejorar el rendimiento o dotar de nuevas funciones al medidor; dejando el campo abierto para retomar este proyecto. Y también se hará una comparativa entre desarrollo con FPGA y microprocesador.

Por último, en los anexos, se incluirá una síntesis de todo lo desarrollado en esta memoria de cara al usuario final del medidor. De tal forma, se podrá utilizar el Medidor de Distancia sin necesidad de comprender su electrónica. Después se comentará el contenido del DVD adjunto a la memoria.

CAPÍTULO 2

DESARROLLO HARDWARE

2. DESARROLLO HARDWARE

El desarrollo hardware del presente proyecto se comprende de:

- Diseño del bloque driver y acondicionador de dispositivos de ultrasonidos (Bloque Auxiliar).

- Utilización de los componentes necesarios en la placa de desarrollo de la FPGA (Bloque Spartan III)

De tal forma, se debe realizar un diseño esquemático y físico de los circuitos necesarios para implementar un driver que actúe sobre un emisor de ultrasonidos y un acondicionador que reciba la señal de un receptor de ultrasonidos.

Paralelamente, se deben explotar los recursos hardware ya diseñados e implementados en la placa de desarrollo Spartan III para hacer que la FPGA funcione adecuadamente y para que el sistema funcione globalmente. Todo ello con las especificaciones que se describirán a lo largo de este capítulo según los objetivos del proyecto.

Se parte con la premisa de que tanto el driver como el acondicionador deben ser alimentados a través de la placa de desarrollo Spartan III. Esto quiere decir que habrá que tener en cuenta las tensiones que suministra dicha placa con el fin de que el sistema final sea un único bloque lo más compacto posible.

Con el mismo objetivo, se deben tener en cuenta los conectores disponibles en dicha placa. También se estudiarán los dispositivos de entradas y salidas que existen en la placa de desarrollo para conseguir llevar a cabo el medidor de distancias con los recursos disponibles.

El sistema final constará de la placa de desarrollo Spartan III y de una placa auxiliar que monte el emisor, receptor, driver y acondicionador. Todo este desarrollo se tratará en profundidad en los siguientes epígrafes.

2.1. DIAGRAMA DE BLOQUES

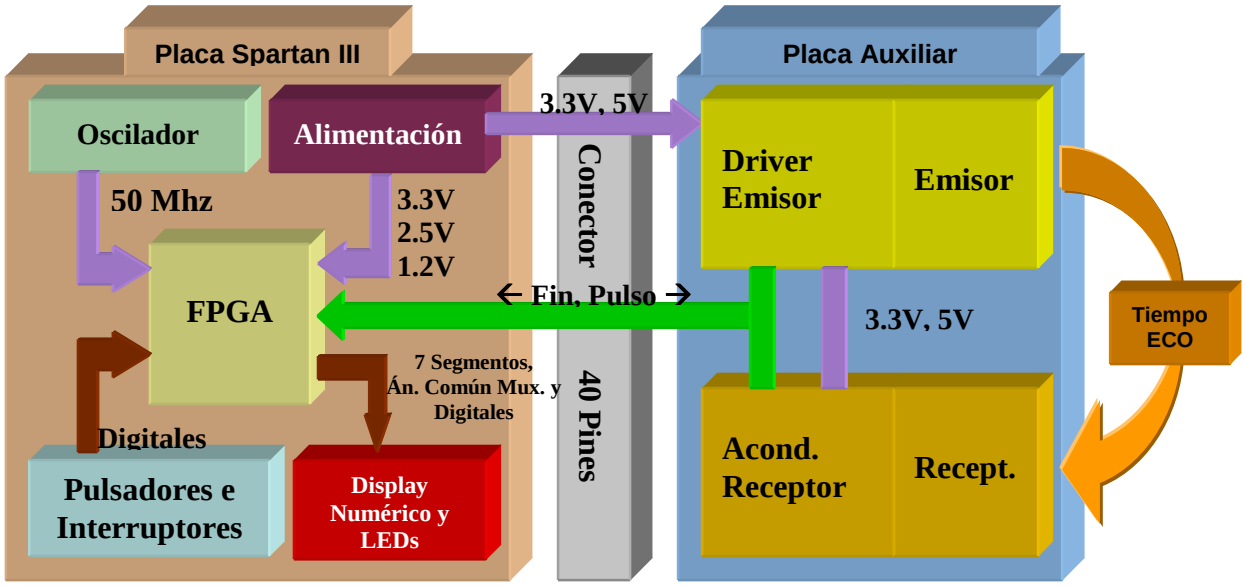


Figura 2.1: Diagrama de bloques Hardware del Medidor de Distancia

En este diagrama (figura 2.1), según se ha definido el sistema, se observan dos bloques funcionales con sus correspondientes subloques. Estos bloques, conectados mediante flechas, conforman el medidor de distancia.

Los colores de las flechas determinan su función. Las conexiones para el acondicionamiento de los circuitos son de color malva. Las conexiones para el control del sistema son de color verde. Las conexiones de color marrón son las de interacción con el usuario.

Tal y como se observa en la figura, el medidor de distancia se compone de dos bloques que se describen a continuación:

Bloque Spartan III

La placa Spartan III dispone del circuito de la FPGA; cuya síntesis dará lugar a un dispositivo que permita conseguir los objetivos mencionados anteriormente. Además de la FPGA, la placa de desarrollo incluye:

- *Reloj:* Este circuito necesitará una frecuencia de reloj para que funcionen todos los circuitos síncronos sintetizados en él; en este caso un oscilador soldado en placa facilitará una frecuencia de 50 MHz.

- *Alimentación:* Existe un módulo de alimentación que suministrará 3.3, 2.5 y 1.2 v. para el funcionamiento de la FPGA. Este módulo aportará

también las tensiones necesarias para los circuitos de la placa auxiliar, que son 3,3 y 5 v.

- *Entrada:* Por otra parte, la placa Spartan III cuenta con varios pulsadores e interruptores directamente conectados a pines de la FPGA. Estos serán utilizados para que el usuario controle el funcionamiento del medidor.

- *Salida:* Dicha placa también dispone de elementos de salida de información hacia el usuario. Los usados para el medidor son un display de 4 módulos de siete segmentos con punto decimal y varios LEDs.

Bloque Auxiliar

En cuanto a la placa auxiliar se aprecia su unión a la placa Spartan III mediante un conector disponible en dicha placa. Por este conector el módulo de alimentación aporta las tensiones ya mencionadas. También se permite la comunicación entre placa auxiliar y FPGA mediante dos señales de control; una de la FPGA hacia la placa auxiliar (Pulso) y otra de la placa auxiliar a la FPGA (Fin).

En esta placa auxiliar se cuenta con un driver que será excitado por la FPGA con la señal Pulso y que hará que el emisor de ultrasonidos, también en esta placa, emita una serie de ondas de ultrasonidos de frecuencia 40 KHz..

Por otra parte, el acondicionador recibirá, a través del receptor, el eco de la onda de ultrasonidos enviada para conseguir obtener la señal digital Fin que se pondrá a nivel alto cuando se este recibiendo una onda de ultrasonidos de frecuencia 40 KHz.

Sin más, se pasa al desarrollo de cada uno de estos bloques funcionales en detalle y tratados más técnicamente.

2.2. PLACA AUXILIAR

Habiéndose descrito el funcionamiento en bloques y las especificaciones de esta placa auxiliar; a continuación se comentará el diseño y funcionamiento de los componentes hardware que se utilizan para su implementación.

2.2.1. Emisor y Receptor de Ultrasonidos

El emisor y receptor de ultrasonidos usados en este proyecto son componentes piezoeléctricos con parámetros muy normalizados. Su misión será la de transformar una señal eléctrica en una onda de ultrasonidos y viceversa.

De tal modo, se puede decir que el emisor de ultrasonidos se comporta como un altavoz y que, al aportarle un tren de pulsos de frecuencia conocida y con la potencia suficiente, emitirá una onda de ultrasonidos de la misma frecuencia y con potencia sonora proporcional a la eléctrica. A esta frecuencia de funcionamiento habitual y muy estándar en este tipo de dispositivos se la llama frecuencia de resonancia del emisor.

Igualmente, el receptor hace las veces de micrófono, es decir, que cuando recibe una onda de ultrasonidos de frecuencia conocida, la transforma en una onda eléctrica semejante al sonido recibido y, por lo tanto, de la misma frecuencia. Por supuesto, esta frecuencia es la de resonancia, en este caso, del receptor.

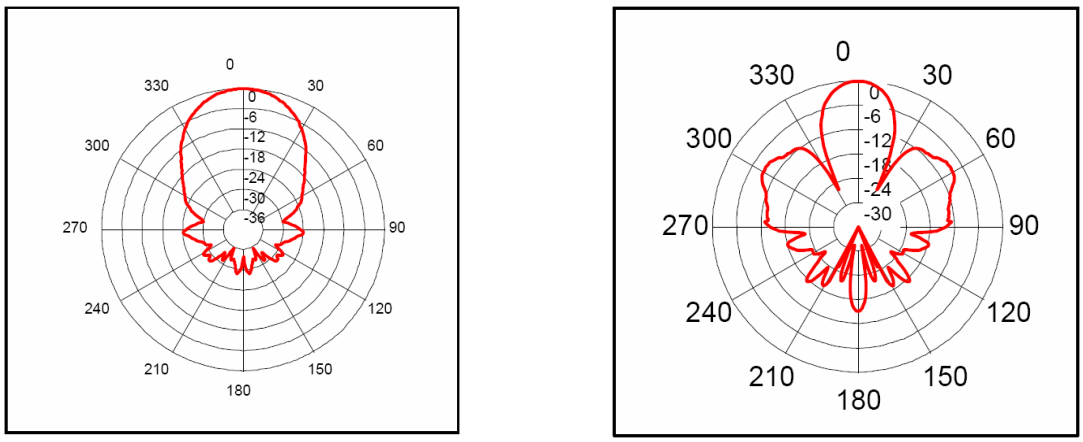
A través del estudio de varios dispositivos emisor y receptor de ultrasonidos (suelen ir emparejados) y de sus hojas de características se han seleccionado los del fabricante *Prowave* y los modelos 400ST160 (Emisor) y 400SR160 (Receptor).

Dado que todos los dispositivos disponibles tenían especificaciones muy parecidas, el principal discriminante para la elección fue la gráfica del ángulo-atenuación de emisión y recepción de ultrasonidos.

Aunque otros dispositivos tienen un haz central más estrecho para que el envío de la onda de ultrasonidos se comporte lo más parecido a una línea recta que sea posible; también tienen haces laterales que, seguramente provocarán el acople de los ultrasonidos entre emisor y receptor sin que sea una verdadera señal de eco.

Sin embargo, los dispositivos elegidos mantienen el compromiso entre un haz central bastante estrecho y la ausencia de haces laterales. El parámetro de ángulo de haz en estos dispositivos para una atenuación de onda de -6 dB. es de 55º.

Todo esto se observa, comparativamente, en la figura 2.2 (obtenida de las hojas de características de estos dispositivos).



400ST/R160 400ET/R180

Figura 2.2: Comparativa de haz de emisión en dispositivos de ultrasonidos

Un parámetro característico ya mencionado es la frecuencia de resonancia. Este parámetro es muy crítico en estos dispositivos. Si se excita el emisor o el receptor recibe una onda de distinta frecuencia a la de resonancia, ninguno de los dos funcionará adecuadamente.

Para el modelo elegido, el 400ST/R160, dicha frecuencia es de 40 KHz. con una frecuencia de corte a -3 dB. de 1 KHz. Esta especificación de frecuencia se menciona en la introducción cuando se describen las ondas de ultrasonidos con las que se trabaja. La caracterización de los dispositivos en frecuencia de resonancia se puede observar en la figura 2.3.:

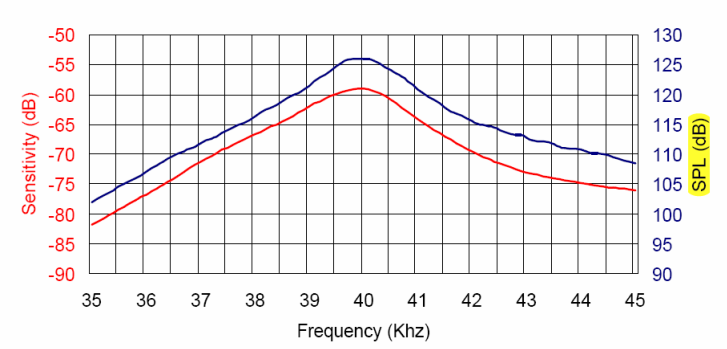


Figura 2.3: Gráfica de sensibilidad y potencia sonora para el par 400ST/R160

En esta figura, obtenida de las hojas de características de los componentes seleccionados (cuyo documento completo se encuentra en el DVD adjunto), se aprecia notablemente el efecto de la frecuencia de resonancia. A medida que la frecuencia de funcionamiento se aleja de los 40 KHz. la atenuación de la señal, en potencia de emisión (Emisor) y sensibilidad (Receptor), es cada vez menor.

Y por último, otro parámetro que conviene conocer, antes de diseñar un driver para el emisor, es la máxima tensión en V*rms*. a la que se puede excitar este dispositivo. Para el emisor elegido, esta tensión es de 20 V*rms*. Puesto que la placa auxiliar y, por tanto, el emisor se alimentarán a una tensión máxima de 5 v., nunca se alcanzará el valor máximo de tensión V*rms*.

En las hojas de características, que se encuentran en el DVD que acompaña a esta memoria, se pueden consultar más parámetros y gráficas, como las de la influencia de temperatura. Sin embargo, debido a las condiciones de funcionamiento del medidor no se hará mayor mención.

Toda esta información será muy relevante a la hora del diseño del driver y acondicionador de estos dispositivos. De tal modo, será tenido en cuenta en los siguientes epígrafes.

2.2.2. Driver del Emisor

Según lo anteriormente comentado, la función de este driver será la de excitar adecuadamente según especificaciones al emisor de ultrasonidos.

Por lo tanto, este driver se dispondrá entre la FPGA y el emisor de ultrasonidos interpretando lo que la FPGA le transmite y adecuándolo en frecuencia y potencia para que el emisor genere una onda de ultrasonidos.

De la FPGA recibirá una señal digital con una tensión a nivel alto de 3,3 v. y 0 v. a nivel bajo. Esta señal se activará a nivel alto durante típicamente 4 milisegundos al comienzo de una medición; posteriormente permanecerá a nivel bajo el resto de la medición. La duración del pulso a nivel alto se fijará mediante la programación de la FPGA según se explicará en el siguiente capítulo.

El emisor de ultrasonidos debe recibir una serie de pulsos a la frecuencia de resonancia anteriormente descrita (40 KHz.). Estos pulsos se producirán únicamente durante el estado a nivel alto de la señal que genera la FPGA.

Este funcionamiento se observa, mediante la representación temporal de las señales de entrada y salida, en la siguiente figura 2.4:

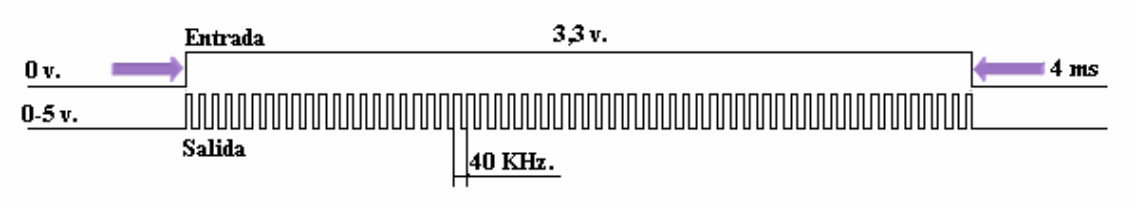


Figura 2.4: Gráfica de funcionamiento del driver del emisor

Según la figura 2.4, este driver genera en su salida un tren de pulsos de 40 KHz. al recibir una señal digital a nivel alto (3,3 v.). Dicho tren de pulsos cesará tan pronto como la señal digital de entrada pase a nivel bajo (0 v.).

La función descrita puede implementarse mediante un circuito astable. De tal modo, para el diseño de driver se elegirá, como elemento principal, un circuito integrado estándar muy adecuado para este tipo de utilización; el LM555.

Este LM555 será capaz de funcionar como un circuito que genera una señal de 40 KHz. y un ciclo de trabajo (D.C.) lo más próximo al 50% sintonizada a través de dos resistencias variables y un condensador.

Para que dicha señal se genere durante el intervalo necesario se actuará sobre el pin de reset del circuito integrado. Este pin de reset actúa a nivel bajo, es decir que, cuando la señal conectada a este pin se encuentre a nivel alto se generará la frecuencia de salida. Mientras que si la señal de entrada se encuentra a nivel bajo, el reset actuará, y no se generará nada a la salida.

Sin embargo, este circuito integrado no tiene suficiente potencia para excitar al emisor de ultrasonidos. Por esto, su salida actuará sobre la base de un transistor de conmutación rápida polarizado entre saturación y corte. Será aquí donde se conecte el sensor de ultrasonidos.

Debido a la frecuencia de conmutación necesaria se elegirá el transistor PN2222 de tipo NPN con un ancho de banda de 300 MHz.

Todo este diseño se muestra en la siguiente figura 2.5.:

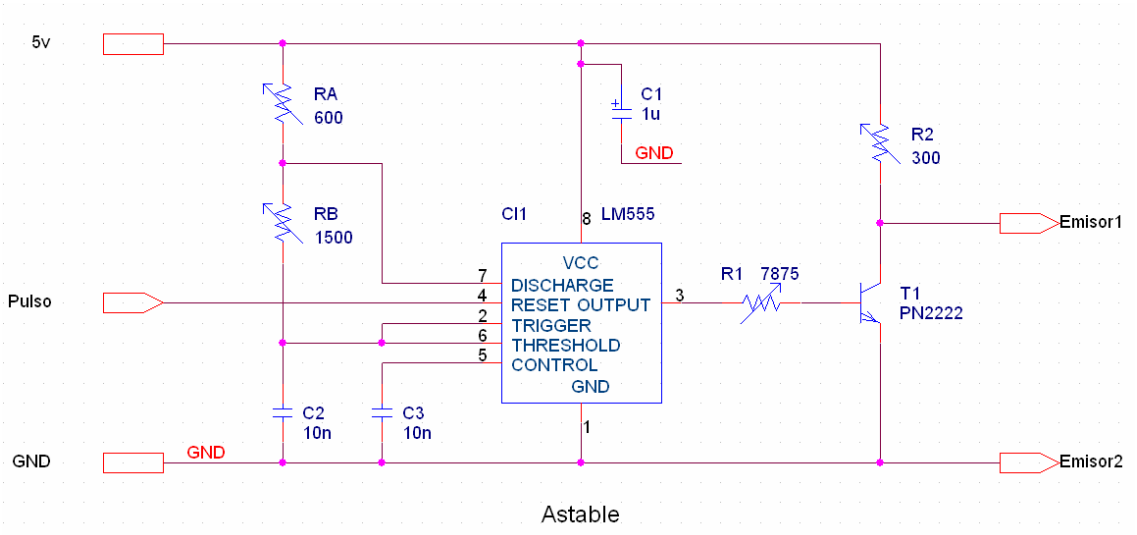


Figura 2.5: Esquema del driver del emisor

En esta figura se aprecian todos los detalles del diseño del driver del emisor de ultrasonidos. Seguidamente se comentarán todos los ajustes y criterios de diseño para la elección de los valores de los componentes electrónicos.

En este circuito, a través de las resistencias R_A y R_B se sintonizará la frecuencia de salida del circuito. Pero al modificarlas también se variará el ciclo de trabajo (D.C.) que debería estar lo más próximo posible al 50%. Todo ello se rige por las ecuaciones dadas por el propio fabricante del LM555; sacadas de las hojas de características que se pueden consultar en el DVD adjunto y que se muestran a continuación:

$$\text{Frecuencia} = \frac{1{,}44}{(R_A + 2R_B)\,C_2} \quad [1] \qquad \text{D.C.} = \frac{R_A + R_B}{R_A + 2R_B} \quad [2]$$

De tal modo, se precisa obtener un D.C. próximo al 50% respetando el valor de la frecuencia de 40 KHz.

Si se fija un valor para R_B de 1500 Ω y para C_2 de 10 nF., sabiendo la frecuencia final y despejando de la ecuación 1 se obtiene un valor para R_A de 600 Ω.

El ciclo de trabajo expresado en esta ecuación 2 es la relación entre el tiempo de la onda a nivel alto y el tiempo total (nivel alto y bajo) de la señal de salida. Esto da como resultado un ciclo de trabajo de un 58% aproximadamente.

Para el condensador C_1 se elige un valor típico de 1 uF. (Para eliminar el efecto de los picos de corriente a alta frecuencia) y para el condensador C_3 se elige un valor de 10 nF.; según las hojas de características de este componente.

Para polarizar el transistor PN2222 entre corte y saturación adecuadamente se jugará con los valores de las resistencias variables R_1 y R_2. De las hojas de características de este transistor se sabe que $V_{CE(SAT)} = 1$ v., $V_{BE(SAT)} = 2$ v. y $h_{FE} \approx 35$.

A través de la aplicación de la ley de ohm a esta etapa de salida se obtienen las siguientes ecuaciones:

$$I_{CE(SAT)} = \frac{V_{CC} - V_{CE(SAT)}}{R_2} \quad [1] \qquad I_{BE(SAT)} = \frac{I_{CE(SAT)}}{h_{FE}} \quad [2]$$

$$V_{BB} = R_1 I_{BE(SAT)} + V_{BE(SAT)} \quad [3] \qquad R_1 = \frac{Vcc - V_{BE(SAT)}}{I_{BE(SAT)}} \quad [4]$$

Si se despeja R_1 en la ecuación 3 se obtiene la ecuación 4, en la cual se conocen todos los datos excepto $I_{BE(SAT)}$. Para hallar este dato se introduce en la ecuación 2 la ecuación 1. Finalmente, con la ecuación 4 ampliada, bastará con asignar un valor a R_2 para obtener el valor correspondiente de R_1 que asegure la saturación del transistor.

Se ha elegido un valor para la resistencia variable R_2 de 300 Ω que fija un valor para la resistencia variable R_1 de 7875 Ω aproximadamente.

No obstante todos estos ajustes serán calibrados experimentalmente, con ayuda de un osciloscopio, en el laboratorio una vez implementado el circuito.

2.2.3. Acondicionador del Receptor

De acuerdo con lo anteriormente comentado, este circuito debe ser capaz de obtener una señal digital a partir del eco que reciba el receptor de ultrasonidos.

Así pues, el acondicionador se localizará entre el receptor de ultrasonidos y la FPGA. De tal forma, tratará la señal eléctrica que obtiene del receptor de ultrasonidos para dar lugar a una señal de niveles lógicos adecuados a la FPGA.

A través del receptor de ultrasonidos recibirá una señal senoidal con amplitud variable y dependiente de la intensidad del eco. Esta señal será, según las especificaciones del receptor y emisor de ultrasonidos y driver del emisor, de frecuencia 40 KHz. No hay que olvidar que, según el principio de sensado de este medidor, la onda de ultrasonidos que llega al receptor debe provenir del eco de la onda que generó anteriormente el emisor.

De cara al circuito de la FPGA se debe generar una señal digital con nivel alto de 3,3 v. y bajo de 0 v. Esta señal se debe poner a nivel alto únicamente durante el tiempo que el receptor de ultrasonidos esté recibiendo ondas ultrasónicas de frecuencia 40 KHz. sea cual sea su amplitud e incluso forma de onda. Esta frecuencia es la de resonancia, idéntica para emisor y receptor.

Por lo tanto, el acondicionador producirá una señal digital que se encuentre a nivel alto (3,3 v.) cuando a su entrada este recibiendo una señal eléctrica de la frecuencia de resonancia (40 KHz.) a través del receptor de ultrasonidos. La señal digital de salida se pondrá a nivel bajo tan pronto como cese la excitación acústica del receptor de ultrasonidos a la frecuencia de resonancia.

El diseño que se propone consta de tres etapas:

1) Etapa amplificadora.

2) Detector de tonos.

3) Adaptador de niveles.

En la etapa amplificadora se debe conseguir aumentar la amplitud de la señal eléctrica obtenida del receptor de ultrasonidos. Esto es necesario porque esta amplitud puede ser del orden de los milivoltios e incluso inferior. Además, de este modo el detector de tonos podrá actuar de forma más fiable.

El circuito integrado que se ha elegido para esta aplicación es el amplificador operacional LM741. El criterio de elección de este circuito es su amplio rango de alimentación de ± 15 v. así como un ancho de banda mínimo de 437 KHz. De este modo está asegurado el funcionamiento del circuito en la placa auxiliar con una alimentación de 5 v. y teniendo que debe amplificar una señal con frecuencia de 40 KHz.

Para el detector de tonos se seleccionó el circuito integrado LM567. Este es un circuito estandarizado para este tipo de aplicaciones. Dicho integrado tiene una alimentación típica de 5 v. El rango de frecuencias de los tonos que puede detectar comprende desde 0,01 Hz. hasta 500 KHz. También tiene un comportamiento muy estable ante variaciones de alimentación, temperatura, ruido o frecuencia de entrada.

Este es el elemento más importante del acondicionador. A través de un oscilador y detectores de frecuencia, pone a nivel lógico bajo su salida (mediante un transistor en colector abierto) cuando detecta una onda en su entrada de la frecuencia sintonizada en el oscilador. Es decir, que realiza la función principal del acondicionador aunque necesite de otros dispositivos para asegurar la fiabilidad.

La última etapa es el adaptador de niveles. Esta consiste en un comparador que evita falsos disparos de la etapa detectora de tonos. Además en esta etapa se invierte la lógica del circuito LM567. Para esta etapa se seleccionó el circuito integrado LM311. Existen tres criterios de selección: su alimentación a 5 v.; su bajo tiempo de respuesta, de 200 nanosegundos; y su salida en colector abierto, que se puede adecuar a otros niveles lógicos.

De tal forma, se ha diseñado un sistema según el siguiente diagrama de bloques:

Figura 2.6: Diagrama de bloques del acondicionador del receptor

A continuación se justificará la elección de los valores de los componentes electrónicos con los que cuenta el circuito de la figura. Se comentarán agrupados por etapas y circuitos al igual que se introdujeron en este epígrafe.

El esquema que corresponde a la etapa amplificadora se muestra a continuación:

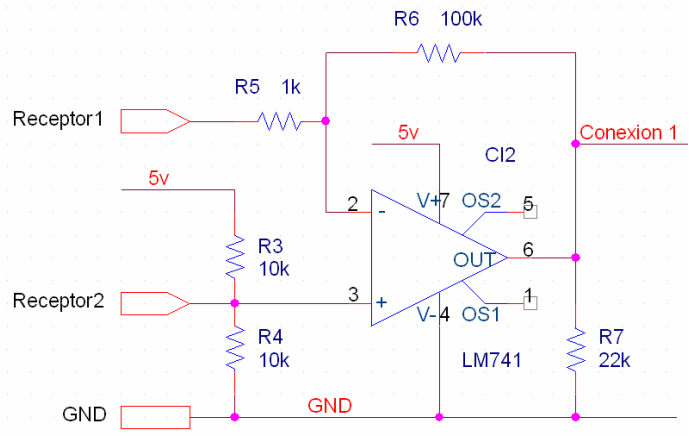


Figura 2.7: Esquema de la etapa amplificadora

En esta figura 2.7 cabe hacer referencia a dos puntos. El primero de estos puntos es un divisor resistivo (R_3 y R_4) que se monta en la entrada de referencia del amplificador. La finalidad es que la salida del amplificador no sature debido a la alimentación no simétrica con la que funciona, generando un offset a la entrada. El segundo punto es el ajuste de la ganancia que se realizará a través de la relación de las resistencias R_5 y R_6. Dada la topología inversora que se ha implementado, la ganancia se define en la siguiente ecuación:

$$G = -\frac{R_6}{R_5}$$

De tal modo, para los valores de las resistencias R_5 de 1 KΩ y R_6 de 100 KΩ, la ganancia de este amplificador es de 100 v/v., suficiente para este tipo de aplicaciones.

Para dimensionar la etapa del detector de tono se consultarán las hojas de características del LM567 que se encuentran en el DVD adjunto. En estas hojas de características se encuentra el detector de tono como una de sus aplicaciones y aporta la ecuación y esquema necesarios para su utilización.

El esquema para esta aplicación se muestra en la siguiente figura 2.8:

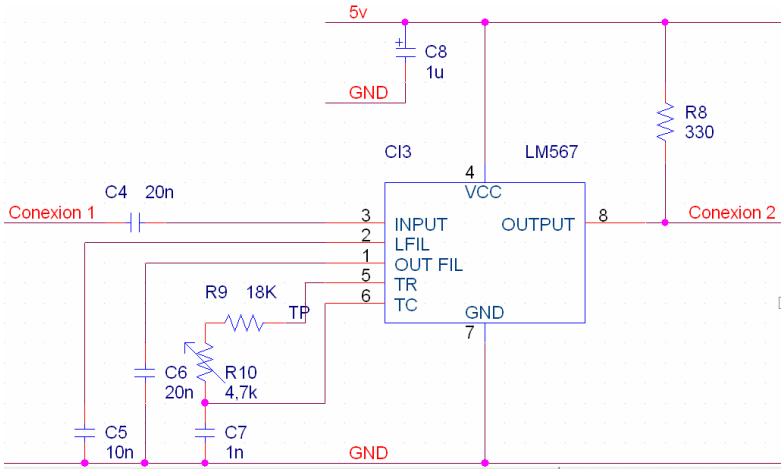


Figura 2.8: Esquema del detector de tonos

En cuanto a lo referente para la sintonización del oscilador con el que comparará el detector de frecuencia se cumple lo siguiente:

$$f_0 \approx \frac{1}{1{,}1\,(R_9 + R_{10})\,C_7}$$

Si se desea obtener una frecuencia de detección f_0 de 40 KHz. y se fija C_7 a 1 nF., se obtiene que la suma de las resistencias R_9 y R_{10} es aproximadamente 22,7 KΩ. De tal forma, la resistencia R_9 será fija y de 18 KΩ mientras que la resistencia variable R_{10} tendrá un valor máximo de 10 KΩ, cubriendo un rango que abarcará desde los 18 hasta los 28 KΩ.

Así la frecuencia de detección (la del oscilador) podrá ser sintonizada experimentalmente en el laboratorio mediante un punto de acceso al pin 5 del LM567 llamado TP según la figura 2.8.

En cuanto al condensador C_5, en las hojas de características se puede encontrar otra ecuación que relaciona su valor con el ancho de banda máximo de la banda de detección. Sin embargo, para el condensador C_5 se ha elegido un valor típico de 10 nF. al igual que el valor típico de C_6 es de 20 nF. y el de C_4 es de 20 nF.

En este caso, también se ha añadido el condensador C_8 para eliminar los efectos negativos de la conmutación del oscilador del LM567 a 40 KHz. Según las hojas de características del componente, un valor adecuado para este condensador es 1 uF.

Dado que la salida de este circuito integrado es de tipo colector abierto, se debe poner una resistencia de pull-up para producir la saturación de este. De este modo, mediante la resistencia R_8 de 330 Ω, se obtendrán los niveles lógicos 0-5 v.

La configuración del adaptador de niveles se ha implementado según la figura 2.9:

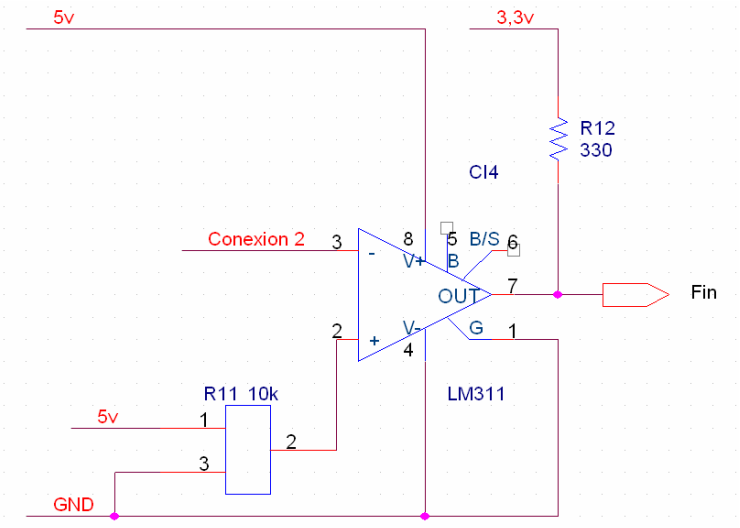


Figura 2.9: Esquema del adaptador de niveles

Según esta figura, en esta etapa simplemente se tiene un divisor resistivo mediante la resistencia variable R_{11} de 10 KΩ para ajustar el valor de comparación con la señal digital de la etapa de detección de tono en el LM311. Este valor se ajustará experimentalmente en el laboratorio, pero siempre será lo más elevado posible para evitar eliminar los falsos disparos y ruido que pueda producir el LM567.

La salida del circuito integrado LM311 es de tipo colector abierto, por lo cual la resistencia R12 de 330 Ω actuará como resistencia de pull-up, polarizando así el transistor de salida en saturación. Dicha resistencia estará conectada a una tensión de 3,3 v., de tal modo, la salida de esta etapa tiene como nivel digital alto 3,3 v. Esta especificación es imprescindible para poder conectarse con la FPGA.

Por último, también conviene comentar la fundamentación de que la conexión de la entrada de la señal digital que proviene de la etapa anterior se conecte al terminal inversor del LM311 y la conexión del valor de referencia se conecte al no inversor. Esto consigue que cuando se detecte la señal de 40 KHz. a la salida del LM311 se tenga un nivel alto de tensión en vez de bajo, especificación dada al principio de este epígrafe.

Con la implementación de este circuito se tendrán cubiertas las especificaciones de este acondicionador del receptor y se podrán detectar frecuencias de 40 KHz. captadas en forma de onda acústica por el receptor de ultrasonidos.

2.2.4. Placa de Circuito Impreso

Ya se han definido y concretado cuales son los circuitos y componentes que conforman la placa auxiliar de este proyecto así como sus valores y las conexiones entre ellos. Este es el punto de partida para la implementación física, en una placa de circuito impreso, de la placa auxiliar.

Las figuras 2.5, 2.7, 2.8 y 2.9 de los epígrafes anteriores recogen todo lo mencionado, incluyendo valores e identificación de componentes. Sin embargo, esos esquemas tienen carencias como el conector hacia la placa Spartan III así como los dispositivos de ultrasonidos, a parte de aunarlo todo en un solo diseño.

Es por esto que, se necesita un esquema más general, que incluya todo lo que abarque la placa auxiliar. Los anteriores esquemas mostrados se han realizado mediante el paquete de diseño electrónico llamado *OrCAD* de la firma *Cadente Design Systems, Inc.* Concretamente con la aplicación llamada *OrCAD Capture.*

Según lo mencionado, se ha realizado un nuevo esquema donde el diseño se abstrae a un nivel superior y tanto el driver como el acondicionador aparecen como bloques. En este nivel solamente se muestran los dispositivos funcionales, sin conocer que es lo que hay en su interior. Este diseño se muestra en la siguiente figura 2.10:

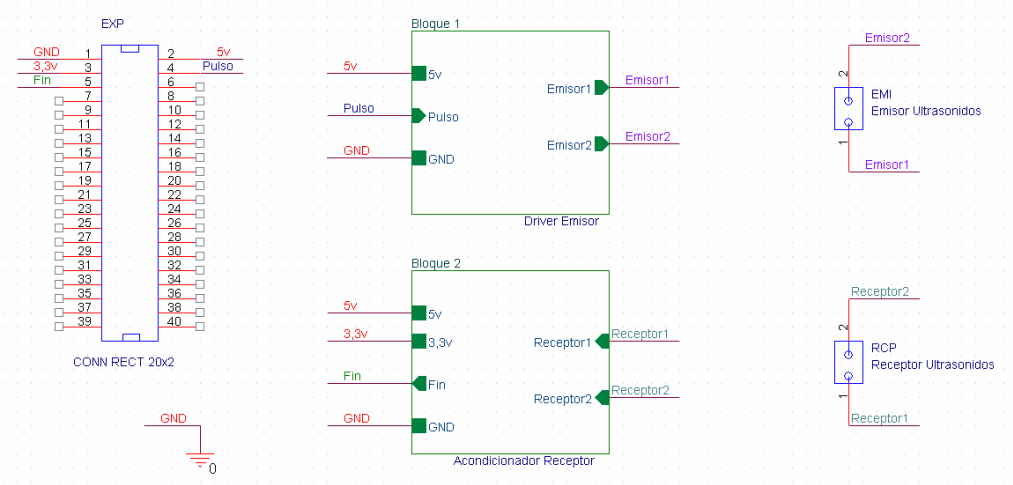


Figura 2.10: Esquema de la Placa Auxiliar

En esta figura se observan las conexiones entre cada uno de los bloques con el conector de 40 pines a la placa Spartan III y con el emisor y receptor de ultrasonidos. Para ello se utilizan alias en los conductores. Los que identifican alimentación son de color rojo y el resto tiene cada uno un color propio para mejor visualización.

A cada uno de los componentes que se han introducido en este esquema así como los del interior de los bloques, que tendrán jerarquizados todos los componentes asociados; se les tiene que asignar lo que se llama huella o footprint del componte.

Esta huella no es más que el desarrollo físico, con las medidas reales, del componente en una placa. Dicho desarrollo debe contar con un espacio donde no se puedan sobreponer componentes, un espacio que es el encapsulado del componente, todos los agujeros necesarios identificados para la soldadura del componente en tantas capas como sea necesario y tantos anillos de cobre como agujeros y capas estén definidos.

Obviamente se presupone la soldadura por tecnología *Through Hole*, es decir, a través de agujero que atraviesa la placa. También se usarán placas de dos capas, superior e inferior. Además hay que tener en cuenta que la mínima separación entre pines de un componente es de 100 *mils*, es decir, de 100 milésimas partes de una pulgada.

Para este diseño se han creado diversas huellas, una para cada encapsulado o forma de componente; independientemente de su función. Estas huellas fueron generadas a través de la aplicación *OrCAD Layout*, a través de la cual se generará la placa. Dichas huellas se apreciarán en el diseño final y por esto no se muestran por separado.

Cabe mencionar que, por supuesto, habrá que hacer una asignación de cada componente con su tipo de huella. Este trabajo se realiza desde OrCAD Capture relacionando en componente con el nombre de la huella que le corresponde.

Seguidamente, ya en OrCAD Layout, se generará un diseño (extensión .max) donde se muestren todos sus componentes y sus conexiones, aún sin rutar. En este momento se tendrá que delimitar el borde de placa y emplazar los componentes.

El emplazamiento de los componentes puede ser de forma automática pero, en este proyecto se decidió hacerlo manualmente para poder decidir la colocación de los dispositivos de ultrasonidos, del conector, y agrupar los componentes según sus funciones.

Una vez se tienen emplazados los componentes, definido el borde de placa y seleccionadas las opciones de rutado; se puede proceder a realizar al rutado automático de la placa. Las pistas de alimentación tendrán un ancho de 20 mils mientras que el resto 15 mils. Además las pistas de la capa inferior suelen ir en vertical mientras que las de la superior en horizontal.

Tras varias iteraciones, analizando los cambios a realizar en el emplazamiento, se consigue que la placa auxiliar quede totalmente definida con todas sus pistas rutadas sin necesidad de implementar vías.

Todo esto da lugar a una placa que cumple todas las especificaciones mencionadas y donde se han tenido en cuenta ciertas normas para evitar las emisiones y perturbaciones electromagnéticas (*EMIs*). Dicho diseño se muestra en la siguiente figura 2.11:

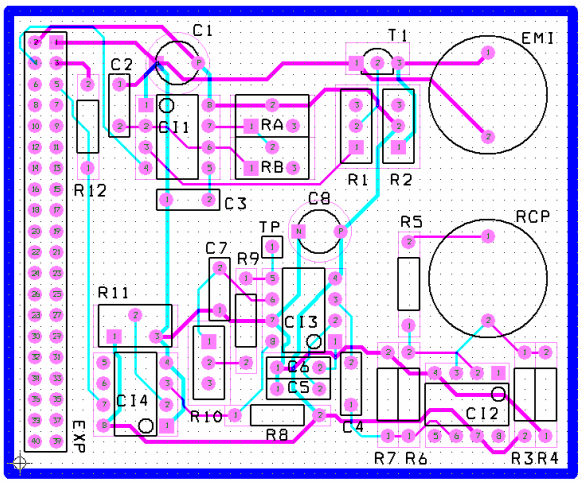


Figura 2.11: Layout de la Placa Auxiliar

Es importante hacer notar que los colores de esta figura se han puesto en negativo para que se aprecie mejor la misma. No obstante, las pistas rosas son las de la capa superior mientras que las azules las de la capa inferior.

Cada uno de los componentes aparece identificado con el mismo nombre con el que aparecía en el esquema correspondiente. Además en algunas huellas, se puede observar la fina línea exterior que evita que varios componentes se superpongan.

También cabe hacer mención a que la línea azul es el borde de placa y que cada uno de los puntos grises de la matriz representa una distancia de 50 mils. Además la figura 2.11 se encuentra en una escala aproximada de 1 : 0,89. Con todo ello se deduce que la medida de la placa auxiliar es de 2650x2200 mils, que supone aproximadamente 67x56 mm.

Esta figura y los archivos que se han generado suponen suficiente información para poder conseguir la fabricación de esta placa de circuito impreso. Todos los archivos generados en cualquiera de las aplicaciones de OrCAD se encuentran en el DVD que acompaña a esta memoria.

El archivo que se debe abrir en OrCAD Capture se llama *Placa Auxiliar.opj*, y el que se debe abrir en OrCAD Layout se llama *Placa Auxiliar-1.max*. Las huellas de componentes que se han generado se encuentran en la librería *Proyecto.llb*.

Una vez se tenga la placa físicamente implementada, solamente quedarán por soldar los componentes de la misma para obtener la placa auxiliar que se necesita para el funcionamiento del medidor de ultrasonidos.

2.3. PLACA DE DESARROLLO Spartan III

A diferencia de la placa auxiliar, la placa de desarrollo *Spartan III* es un producto comercial. A cargo dicha fabricación se encuentra la firma *Digilent*, en colaboración con *Xilinx*. Estos fabricantes han diseñado un sistema para permitir realizar un desarrollo y test de FPGAs completo de manera muy sencilla y versátil, lo que reduce el tiempo de desarrollo.

De tal forma, esta placa cuenta con todo el Hardware necesario para sintetizar un circuito en la FPGA y probarlo con todos sus entrenadores y dispositivos de salida. Pero también, como es el caso, es posible realizar un prototipo de un circuito más complejo que incluso integre más Hardware.

La principal ventaja de esta placa es su diseño compacto que incluye multitud de funciones y dispositivos. De este modo, se reduce el tiempo de desarrollo de un prototipo únicamente para acondicionar la FPGA en cuanto a frecuencia de reloj y alimentación, así como para interactuar con dicha FPGA.

Por lo tanto, para alcanzar los objetivos anteriormente descritos se debe estudiar la disponibilidad, funcionamiento y conexión de los dispositivos de la placa a la FPGA mediante sus hojas de características, presentes en el DVD adjunto. Seguidamente habrá que conectar las señales del circuito sintetizado en la FPGA a los recursos necesarios de la placa mediante la asignación de pines.

Sin embargo, en este epígrafe se detallará, en primer lugar, la FPGA montada en la placa Spartan III así como sus características. Posteriormente, se expondrán a grandes rasgos los dispositivos y recursos de los que dispone la placa de desarrollo. Por último se describirán exhaustivamente los dispositivos implicados directamente en el funcionamiento del medidor de distancias.

2.3.1. Descripción de la FPGA *Spartan 3*

La FPGA (*Field Programable Gate Array*) es un dispositivo electrónico estándar formado por una gran cantidad de puertas lógicas y otros elementos. La disposición de estas puertas lógicas y de los bloques que conforman permiten sintetizar multitud de circuitos digitales sin tener que cambiar de Hardware. Esto es posible gracias a la posibilidad de configurar dichos bloques mediante programación.

Estos circuitos son cada vez más usados en el desarrollo de nuevos sistemas electrónicos. Esto es debido a la facilidad de prototipado que permiten estos dispositivos programables. Todo esto aporta una capacidad de integrar multitud de circuitos digitales en un mismo chip permaneciendo a su vez el contenido totalmente opaco al exterior.

Dichas cualidades hacen que las FPGAs sean soluciones muy económicas, compactas y con bajos tiempos de desarrollo y permiten que, cada vez más, sustituyan a circuitos como los microprocesadores.

La FPGA que se monta en la placa de desarrollo Spartan III es de la familia de FPGAs *Spartan 3* de Xilinx.

A continuación se explicará exhaustivamente todo lo referente a esta familia de componentes electrónicos y al modelo concreto utilizado en este proyecto. Por lo tanto, en primer lugar, se describirá la estructura interna de la FPGA. Seguidamente se estará en disposición de comprender la configuración y programación de la misma. Y, por último, se expondrán detalladamente las especificaciones y características FPGA con la que cuenta la placa de desarrollo Spartan III.

Para el funcionamiento de la FPGA se cuenta con una estructura característica que se describirá a continuación. La estructura interna de la familia de FPGAs Spartan 3 se compone de cinco bloques funcionales distintos, que son los siguientes:

- CLB's (*Configurable Logic Blocks*): Como su nombre indica, son bloques lógicos configurables. Estos bloques contienen tablas basadas en memoria RAM (LUT's) que les permite implementar la lógica y registros necesarios para sintetizar flip-flops o latches. De tal forma, estos bloques se usan para generar un gran número de funciones lógicas así como almacenamientos de datos.

- IOB's (*Input/Output Blocks*): Estos bloques son los que establecen el control del flujo de datos entre los pines de entrada/salida de la FPGA y la lógica interna de la misma. Dichos bloques tienen multitud de funcionalidades que, en el nivel de diseño al que se encuadra este proyecto, no son relevantes.

- RAM Blocks: Son bloques de memoria RAM de 18 Kbits divididos en dos subloques cada uno. Por supuesto, estos bloques son los que se encargan de la mayor parte del almacenamiento de los datos.

- Multipliers Blocks: Estos bloques son multiplicadores dedicados que calculan productos a través de dos entradas de datos de 18 bits cada una.

- Digital Clock Manager (*DCM*) Blocks: Estos bloques permiten un control total en la generación de todo tipo de señales de reloj a través de diversas configuraciones y a partir de una frecuencia base de entrada.

Estos bloques se distribuyen en el interior del encapsulado de la FPGA según la figura 2.12:

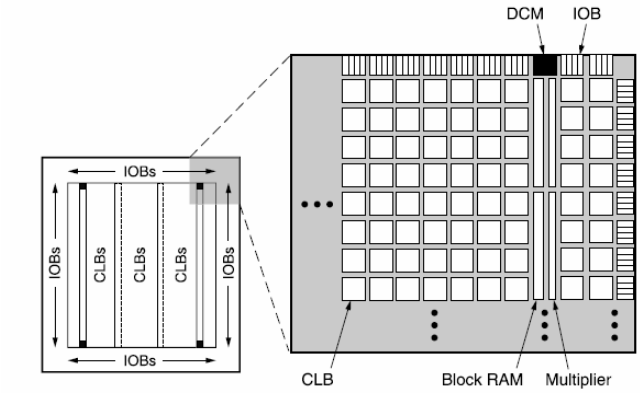


Figura 2.12: Estructura interna de la FPGA XC3S200

Como se aprecia en la figura, el producto final es una matriz de diversos bloques funcionales. Pero para el correcto funcionamiento y síntesis de circuitos en la FPGA es necesario que dichos bloques se conecten entre sí. De está conexión se encarga una matriz de interruptores asociada a cada bloque que actúa sobre una extensa red de conductores. Así, es posible realizar múltiples conexiones y rutados entre bloques sobre el mismo Hardware, dependiendo de los valores programados sobre la matriz de interruptores.

La programación de estos circuitos, significa por ende, la configuración del sistema. Dicha configuración debe cargarse en la FPGA cada vez que se alimenta. Esto se realiza mediante la escritura de la configuración en memoria volátil a través de la programación por el cable JTAG que comunica computadora y FPGA. Aunque dicha configuración también puede almacenarse en un dispositivo no volátil, como la memoria Flash Xilinx XCF02S con la que cuenta la placa Spartan III, y ser cargada automáticamente cada vez que se reinicia. A esta posibilidad de configurar los interruptores a través de programación volátil por JTAG o no volátil en memoria Flash se le denomina Modo de Configuración.

Habiéndose descrito ya a qué se debe la flexibilidad de las FPGAs Spartan 3 de Xilinx, así como los Modos de Configuración de estos dispositivos en la placa Spartan III; se describirán a continuación las características de la FPGA usada para el diseño e implementación del medidor de distancias.

Para caracterizar la FPGA montada en la placa Spartan III que se utiliza en este proyecto se mostrará seguidamente un esquema (obtenido de las hojas de características de la placa, DVD adjunto) donde se explica cada uno de los campos de la denominación de la misma:

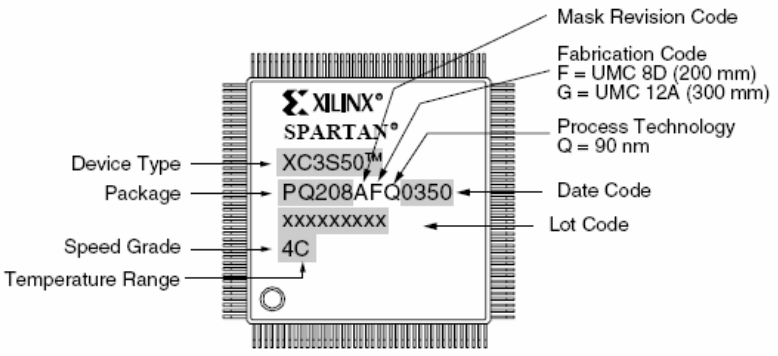


Figura 2.13: Esquema de identificación de FPGA

La FPGA que se utiliza en este proyecto es la que tiene el siguiente marcado en el encapsulado:

XC3S200
FT256AFQxxxx
xxxxxxxx
4C

Esto significa, según la figura 2.13 y las hojas de características de la placa Spartan III, que la FPGA utilizada tiene, como características más relevantes para este proyecto, las que se exponen en la tabla 2.1:

Características de la FPGA

Alimentación	Símbolo	Valor
Interna	V_{CCINT}	1,2 v.
Auxiliar	V_{CCAUX}	2,5 v.
Drivers Salidas	V_{CCO}	3,3 v.

Modelo	Puertas Lógicas	Celdas Lógicas
XC3S200	200,000	4,320

Encapsulado	# Pines	Tipo de Pines
FT256	256	FTBGA(1)

Rango Temperatura	Valor Mínimo	Valor Máximo
C → Comercial	0 ºC	80 ºC

Notas:
3. Fine-Pitch Thin Ball Grid Array

Tabla 2.1: Especificaciones de Xilinx sobre la FPGA XC3S200

Con los datos aportados acerca del funcionamiento, estructura, configuración y características de la FPGA de la familia Spartan 3 montada sobre la placa de desarrollo Spartan III; se concluye la descripción de dicho circuito desde el punto de vista Hardware.

2.3.2. Descripción de la Placa Spartan III

Como se ha comentado anteriormente, el diseño del medidor engloba una placa comercial de desarrollo, la placa de desarrollo Spartan III. A continuación se presentarán las características más relevantes para el desarrollo de este proyecto.

Los dispositivos y recursos con los que cuenta la placa de desarrollo Spartan III son variados y de diversa naturaleza. A parte de la FPGA [1], los recursos con los que cuenta la placa Spartan III se pueden apreciar en la figura 2.14:

Figura 2.14: Recursos de la placa Spartan III

A continuación se ubicarán físicamente en la placa de desarrollo cada uno de los puntos a los que hace referencia la figura 2.14. Para ello se ha de tener en cuenta el número que aparece entre corchetes en cada uno de los

recursos de esta figura. Este número coincidirá con el que se encuentra identificando a cada uno de los recursos de las siguientes figuras 2.15 y 2.16:

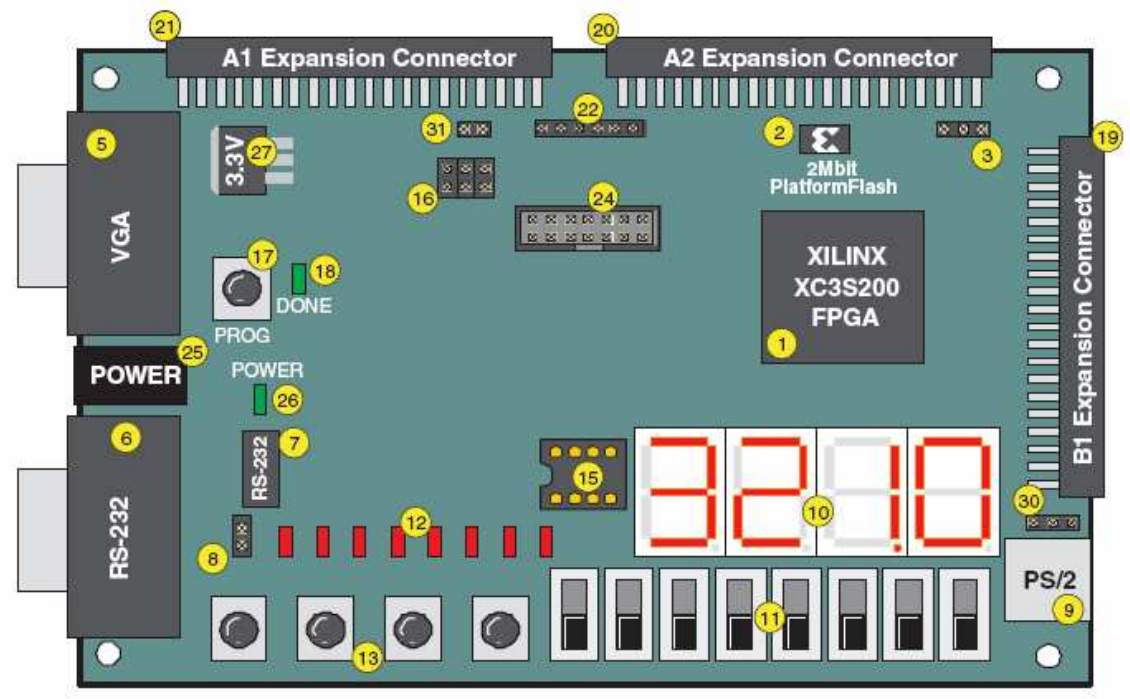


Figura 2.15: Localización de recursos en el frontal de la placa Spartan III

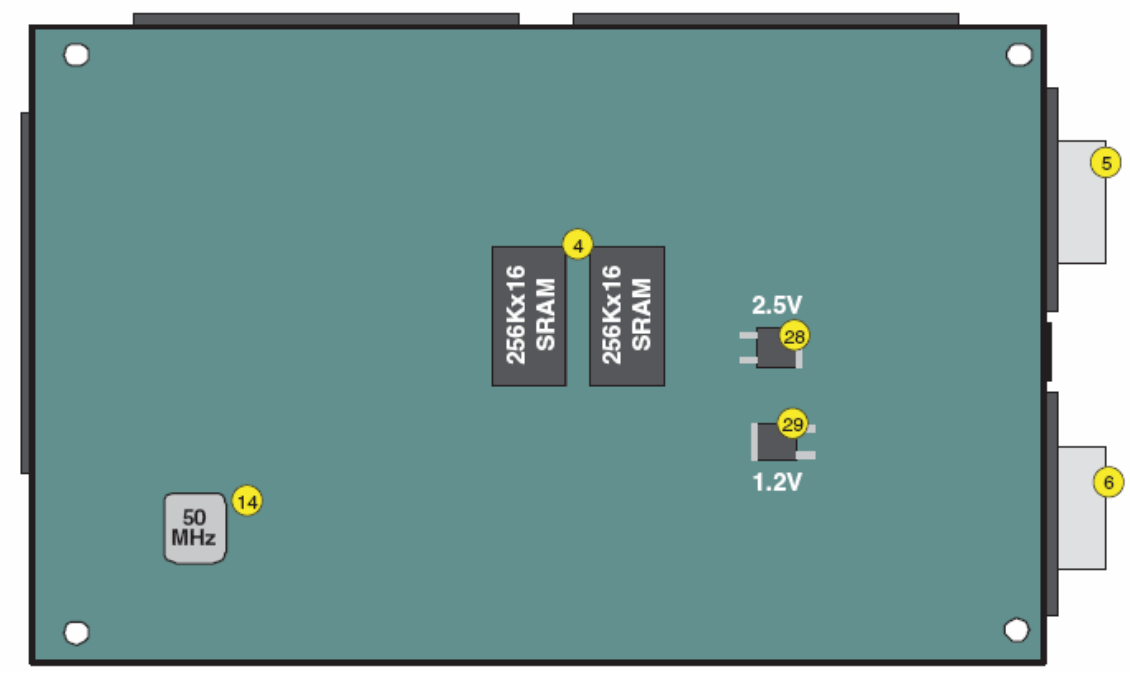


Figura 2.16: Localización de recursos en reverso de la placa Spartan III

Una vez enumerados y localizados todos los recursos de los que está dotada la placa y quedando introducido todo el Hardware; se proseguirá con otra breve enumeración de lo que se requiere de ella para poder implementar el medidor de distancias.

Según lo mostrado en el diagrama de bloques Hardware de la figura 2.1 y otras consideraciones mencionadas a lo largo de este capítulo; todas las necesidades y condiciones que ha de cumplir esta placa y los motivos que llevan a su elección, son los siguientes:

- Aportar un sistema mínimo (alimentación y frecuencia de reloj) tanto la FPGA como el circuito de la placa auxiliar.✓
- Poseer dispositivos de entrada para controlar el medidor, que serán pulsadores *(Push Button)* e interruptores *(Switch)*.✓
- Poseer dispositivos de salida para mostrar el resultado de la medida *(Display 7 segmentos)* y el estado del medidor *(LED)*.✓
- Poseer puertos de expansión para poder conectar la placa auxiliar.✓
- Dar soporte y consistencia a todo el conjunto del medidor.✓
- Poseer un potente entorno de desarrollo versátil, funcional y flexible.✓

Como se observa, todos estos requerimientos son cubiertos por la placa de desarrollo Spartan III y, es más, no todos los recursos de esta placa son usados por el medidor de distancias.

Por lo tanto, a continuación se estudia cómo se identifican los pines de la FPGA utilizada. Seguidamente se introduce cuales de estos pines están físicamente conectados a cada uno de los recursos de la placa de desarrollo.

Este conocimiento permitirá posteriormente saber como se debe de programar la FPGA para poder interactuar con dichos elementos. Este paso será una simple asignación que conectará las señales del circuito sintetizado en la FPGA a cada uno de los pines de la misma. Cada pin tiene conectado un recurso que se asignará al circuito sintetizado o no en función de las especificaciones del mismo.

Así se pasa comentar la numeración de los pines de la FPGA. Cuando se describió el encapsulado de la misma en el epígrafe anterior se comentó que dicho circuito tiene un encapsulado que le permitía tener 256 pines dispuestos en forma de matriz de bolas. Esto se puede observar en la figura 2.17:

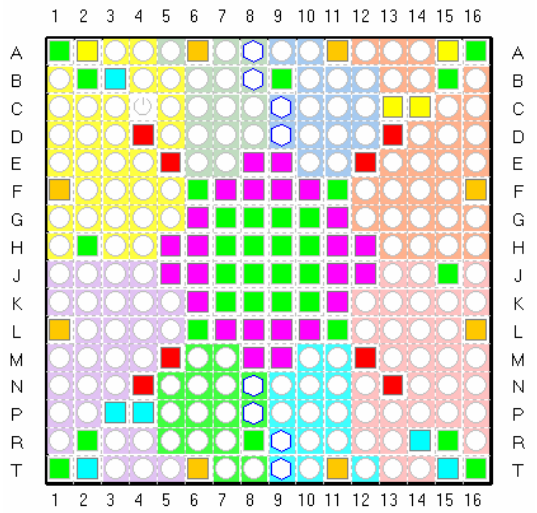


Figura 2.17: Numeración de pines en la FPGA XC3S200

Como se aprecia en esta figura, para numerar inequívocamente cada pin se dividen las líneas de pines en filas y columnas; cada fila tendrá una letra del abecedario asignada de arriba a abajo y cada columna un número asignado de izquierda a derecha. El punto del encapsulado se encuentra en la parte inferior izquierda de la figura.

La mayor parte de estos pines son configurables para conectarse al circuito sintetizado en la FPGA o no; esto es algo que se verá en el siguiente capítulo. En cuanto a esto, conviene comentar que los pines cuadrados y hexagonales no son configurables y que cumplen funciones de acondicionamiento o programación según el siguiente código de colores:

- Verde: Masa.
- Rosa: V_{CCO} (3,3 v.)
- Naranja: V_{CCAUX} (2,5 v.)
- Rojo: V_{CCINT} (1,2 v.)
- Cian y Amarillo: Programación.
- Hexágono: Reloj.

En este momento se está en disposición de definir las especificaciones de los elementos de la placa Spartan III utilizados en el desarrollo del medidor de distancias.

Se comenzará por el acondicionamiento de la FPGA. En lo referente a la alimentación a 3.3, 2.5 y 1.2 v. de esta FPGA, no es necesario tener ninguna consideración especial ya que los pines de alimentación de este circuito son fijos y por lo tanto no se pueden variar ni asignar. En cuanto al oscilador de 50 MHz. cabe mencionar, que la frecuencia de oscilación llega al pin T9 de la FPGA. Con esto será suficiente para que la FPGA quede alimentada y posea una frecuencia de reloj de manera estable gracias a los dispositivos de la placa Spartan.

En cuanto a los pulsadores, cabe mencionar que cuando son accionados ponen a nivel alto (3,3 v.) la señal que llega a la FPGA. Mientras tanto, los interruptores ponen la señal que llevan a la FPGA a nivel alto cuando se acciona su palanca hacia el bloque del display. Estos pulsadores e interruptores llevan, cada uno, su señal a un pin distinto de la FPGA. Esta relación se especifica en las tablas sacadas del Manual de Usuario de la placa (DVD adjunto); que son la 2.2 para los pulsadores y la 2.3 para los interruptores:

Push Button	BTN3 (User Reset)	BTN2	BTN1	BTN0
FPGA Pin	L14	L13	M14	M13

Tabla 2.2: Conexionado de pulsadores a pines de la FPGA

Switch	SW7	SW6	SW5	SW4	SW3	SW2	SW1	SW0
FPGA Pin	K13	K14	J13	J14	H13	H14	G12	F12

Tabla 2.3: Conexionado de interruptores a pines de la FPGA

Los LEDs están polarizados de tal forma que, cuando una señal de la FPGA que llega hasta ellos se pone a nivel alto (3,3 v.), se encienden. Las conexiones entre dichos LEDs y la FPGA se especifica en la tabla 2.4, obtenida del manual de usuario de la placa:

LED	LD7	LD6	LD5	LD4	LD3	LD2	LD1	LD0
FPGA Pin	P11	P12	N12	P13	N14	L12	P14	K12

Tabla 2.4: Conexionado de LEDs a pines de la FPGA

El dispositivo de salida más importante del medidor de distancias es el display de 4 dígitos y siete segmentos por dígito. En él se muestra el resultado procedente de la medida realizada por este sistema. El conexionado de este display no es trivial y requiere de cierto control por parte de la FPGA.

Los cátodos de cada uno de los LEDs correspondientes a los segmentos del display se conectan a la FPGA mediante un bus de 8 bits, incluyendo el punto decimal. Las líneas del bus no se conectan a un solo cátodo; si no que conectan a todos los cátodos de distintos dígitos pero siempre del mismo segmento. En cuanto a los ánodos, son comunes a todos los LEDs de cada dígito y son independientes entre sí.

A esta conexión y funcionamiento se le denomina multiplexión temporal de caracteres en un display y es comúnmente utilizada. Este procedimiento requiere un mayor control por parte del circuito sintetizado en la FPGA pero conlleva un ahorro de espacio de rutado en placa así como de pines de la FPGA.

De tal modo que para dibujar un número en un dígito, se activaría el ánodo correspondiente a dicho dígito y, al mismo tiempo, se introduciría por el bus el número a dibujar en código de 7 segmentos. Tanto los ánodos como cada uno de los cátodos se activan mediante un nivel bajo proveniente de la FPGA.

Para aclarar esta explicación se muestra la figura 2.18:

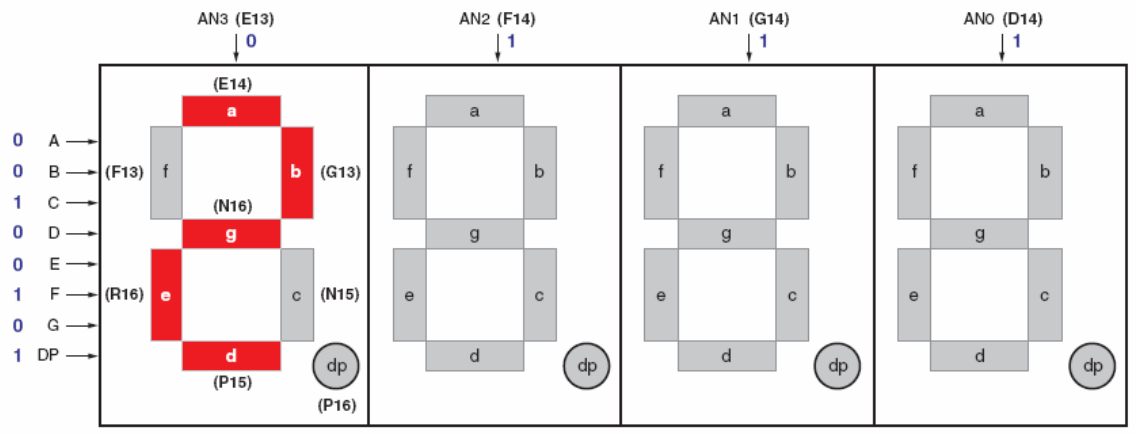


Figura 2.18: Control del display de 4 dígitos de la placa Spartan III

En la figura 2.18 también se especifica el pin de la FPGA, común a los 4 dígitos, correspondiente a cada uno de los cátodos de los segmentos; así como el pin de la FPGA de cada uno de los ánodos de los dígitos.

Para la conexión a la placa auxiliar se usará el puerto de expansión A2 de la placa Spartan III. Esta elección se realiza debido a que la unión mediante este conector de 40 pines dará mayor rigidez al conjunto, a que las señales de este conector no tienen ningún tratamiento especial, como lo tienen las del puerto serie, y a que están accesibles las alimentaciones de 5 y 3.3 v. necesarias para el funcionamiento de la placa auxiliar. Los otros dos puertos de expansión son también viables pero destinan la mayor parte de sus señales a la programación de la FPGA; en vez de a entradas y salidas para el usuario.

A lo largo del desarrollo Hardware de la placa auxiliar se han especificado las señales y alimentación que compartiría con la placa Spartan III. Estas señales son la de Pulso (0,1 → 0, 3.3 v. de salida de la FPGA) y la de Fin (0,1 → 0, 3.3 v. de entrada a la FPGA). En cuanto a las tensiones de alimentación, se precisan 3.3 y 5 v. El conector de expansión A2 permite realizar estas conexiones entre FPGA y placa auxiliar. El conexionado de este expansor a los pines de la FPGA y a la alimentación se muestra en la tabla 2.5:

Schematic Name	FPGA Pin	Connector		FPGA Pin	Schematic Name
GND		1	2		VU (+5V)
V_{CCO} (+3.3V)	V_{CCO} (all banks)	3	4	(E6)	PA-IO1
PA-IO2	(D5)	5	6	(C5)	PA-IO3

Tabla 2.5: Conexionado del expansor A2 a pines de la FPGA

En esta tabla solo se muestran los seis primeros pines del conector de expansión. Este conector tiene 40 pines, sin embargo, con 5 pines será suficiente para los requerimientos de la placa auxiliar, anteriormente expuestos.

Pero, es necesario conocer la ubicación de dichos pines en el conector de expansión A2. Esto se aclara en la figura 2.19 que se muestra a continuación:

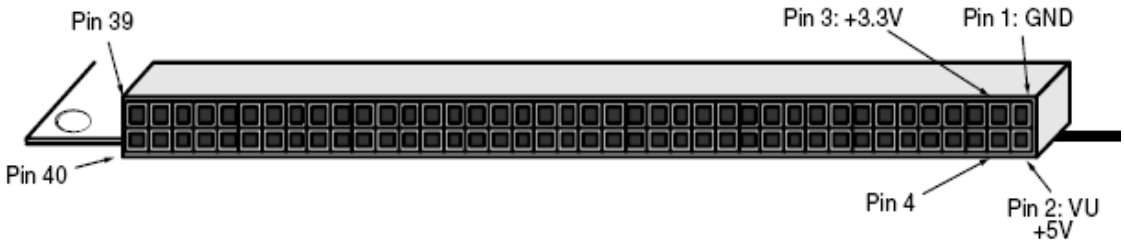


Figura 2.19: Numeración de pines del expansor A2 de la placa Spartan III

Esta información es imprescindible para el diseño de la placa de circuito impreso correspondiente a la placa auxiliar. Como se comentó en el epígrafe

correspondiente a este diseño, las especificaciones de este conector se irían justificando a lo largo del desarrollo de la memoria.

Por último, en cuanto a la versatilidad de la que dota a la FPGA la placa Spartan III, se explicará como se realiza la programación de dicha FPGA.

En el epígrafe anterior se explicó el funcionamiento y los modos de configuración de la FPGA y en este epígrafe se ha mostrado mediante figuras la ubicación del puerto JTAG de la placa Spartan III. Esto será útil para comprender la programación de la FPGA.

Mediante este puerto y un cable JTAG con conexión a puerto paralelo de PC suministrado por el fabricante, se transmitirá la información de la programación del PC a la FPGA a través de un entorno de desarrollo (que se explicará en el siguiente capítulo).

Sin embargo se comentó anteriormente que dicha programación se podría realizar sobre memoria volátil o sobre la memoria Flash XCF02S que hay en la placa Spartan III y cuya ubicación se ha especificado en este epígrafe. A estas diversas programaciones se las denomina modos de configuración.

Dicho modo dependerá del estado de dos grupos de jumpers de la placa Spartan III. Estos son los jumpers J8 y JP1 que se encuentran en la parte superior central y derecha de la placa, respectivamente.

Según la tabla 2.6, que aparece en el Manual de Usuario de la placa (que se encuentra en el DVD adjunto) se trabajará con dos configuraciones que serán las siguientes:

Configuration Mode <M0:M1:M2>	Header J8 Settings	Jumper JP1 Setting	Description
Master Serial <0:0:0>			DEFAULT. The FPGA automatically boots from the Platform Flash.
			The FPGA attempts to boot from a serial configuration source attached to either expansion connector A2 or B1.
JTAG <1:0:1>			The FPGA waits for configuration via the four-wire JTAG interface.

Tabla 2.6: Selección de Modo de Configuración de FPGA en la placa Spartan III

De tal modo, para configurar la FPGA mediante el cable de programación JTAG y probar un diseño sin que este se ejecute cada vez que se alimente a la placa se dispondrán los jumpers como se indica en el Modo de Configuración JTAG. Sin embargo, cuando se quiera que el diseño definitivo no se pierda al quitar la alimentación de la placa, se utilizará el Modo de Configuración Master Serial de tipo DEFAULT.

Todos los recursos que se han comentado en mayor detalle, serán de uso común en la etapa de diseño e implementación del medidor de distancias mediante la placa Spartan III y su FPGA.

Con toda la información aportada acerca del desarrollo Hardware que se ha realizado sobre la placa auxiliar y de los recursos Hardware de la FPGA y placa Spartan III; se está en disposición para abordar adecuadamente el diseño del circuito implementado en el interior de la FPGA. Esto se trata en el siguiente capítulo.

CAPÍTULO 3

DESARROLLO VHDL

3. DESARROLLO VHDL

El desarrollo VHDL de este proyecto definirá el código VHDL necesario para hacer que la FPGA permita al sistema final funcionar según las especificaciones descritas. Esto engloba tanto la interfase con el usuario, como la obtención y tratamiento de datos.

Para centrar el desarrollo realizado sobre la FPGA en código VHDL, primeramente se introducirán las especificaciones y objetivos que debe cumplir el circuito sintetizado en dicho dispositivo. No obstante, se comentará el funcionamiento general de la FPGA como caja negra y se describirán en profundidad los requerimientos que debe cumplir el circuito de la FPGA.

Seguidamente y para comprender la totalidad del desarrollo VHDL realizado, en este capítulo se comentarán los principios de este lenguaje, ventajas, características, así como la estructura básica que se ha de respetar.

También se comentará el funcionamiento del entorno de desarrollo utilizado para la programación de la FPGA. Se explicará el modo de uso del mismo según sus funcionalidades partiendo de la creación de un proyecto hasta llegar a la síntesis de un código y su programación en la FPGA. Sólo de este modo quedará totalmente definido el proceso de programación en la placa de desarrollo Spartan III.

Por último y con toda esta información, se pasará a la descripción exhaustiva de cada uno de los bloques del código VHDL desarrollado para esta aplicación; explicando y comentando, paso a paso, su funcionamiento.

Todo este contenido será desarrollado en mayor detalle en los siguientes epígrafes de esta memoria.

3.1. REQUISITOS DE LA FPGA

En el Capítulo Segundo de esta memoria se presentó un diagrama de bloques que muestra todo lo que se refiere al hardware de este proyecto. De esta forma, se introdujo el papel que la FPGA y la placa de desarrollo deberían desempeñar en el funcionamiento del medidor.

Posteriormente también se comentaron los recursos hardware de la placa de desarrollo Spartan III y de qué forma se utilizarían en este proyecto para poder obtener el resultado mencionado.

De este modo, el circuito codificado en VHDL será capaz de explotar los recursos de la placa de desarrollo de forma adecuada así como de realizar las funciones necesarias para la consecución del medidor de ultrasonidos.

Según lo comentado y en concordancia con la figura 2.1 del Capítulo Segundo de la presente memoria, el circuito sintetizado en la FPGA y los requisitos que debe cumplir finalmente la misma, responden a la siguiente figura 3.1:

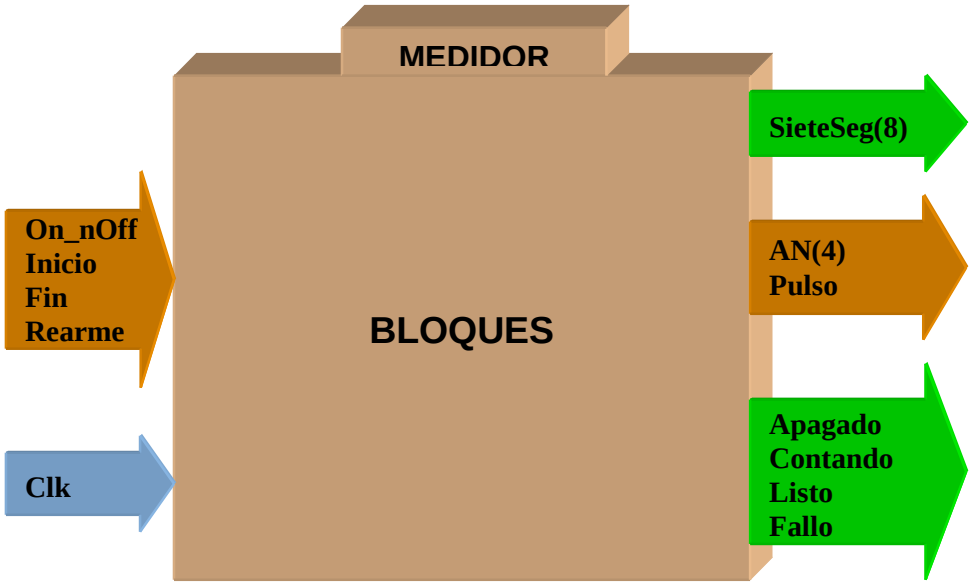


Figura 3.1: Señales de entrada y salida del circuito Medidor

En esta figura se observa un gran bloque en forma de caja negra del cual entran y salen señales. Esta caja negra no es más que el código VHDL de mayor nivel en la jerarquía del código generado.

El nombre de la entidad y por lo tanto del circuito, es *Medidor* con la arquitectura llamada *Bloques* (debido a la estructura altamente jerarquizada que posee el diseño). Todo este código se encuentra en el archivo *Medidor.vhd* que se comentará en el siguiente epígrafe.

Dado que esta es la estructura VHDL de mayor nivel, este circuito se puede tomar como el propio encapsulado de la FPGA, utilizando los pines adecuados. Es por esto que se observa, a nivel funcional, cómo señales que en la figura 2.1 del Capítulo Segundo entraban y salían de la FPGA, ahora están relacionadas con este circuito.

De tal forma y como se observa en la figura 3.1, las señales de entrada al Medidor son: On_nOff, Inicio, Fin, Rearme y Clk. Mientras tanto las salidas son: SieteSeg(7), AN(4), Pulso, Apagado, Contando, Listo y Fallo. El código de colores de las flechas es el siguiente: señales en flecha azul son señales de sincronismo, señales en flecha marrón son señales de control y señales en flecha verde son señales de datos e indicadores de estado.

Las señales de entrada y salida que sirven de interfase con el usuario son las siguientes:

- *On_nOff:* Esta señal vendrá de un interruptor de la placa de desarrollo y servirá para encender el sistema o dejarlo en standby señalizando con un LED de la placa.

- *Rearme:* Es un pulsador de la placa de desarrollo destinado a reiniciar el sistema de control, para sacar al sistema de estados indeseados o pérdidas de control, si los hubiera.

- *Inicio:* Esta señal, conectada a un interruptor, será la que permita la medición continua o la medición con datos aislados. Con ella se indica cuando se toman medidas y cuando no.

- *AN(4):* Este vector, que realmente está compuesto de 4 señales independientes, es una de las señales más importantes. Mediante estas líneas de control se pueden mostrar datos en el display de 4 módulos de 7 segmentos multiplexados en el tiempo tal cual requiere la placa de desarrollo Spartan III.

- *SieteSeg(7):* Este es el vector que contiene los datos en código 7 segmentos que se van dibujando en el display con el orden que requiere la operación de refrescado y totalmente sincronizados con el vector AN(4).

- *Apagado:* Esta señal, conectada a un LED, indica cuando la señal On_nOff está a nivel alto o bajo. Así, cuando la mencionada señal está a nivel bajo el LED estará encendido y viceversa.

- *Contando:* Esta señal, que se conecta a un LED, se activa al emitir una onda de ultrasonidos y queda así hasta que finaliza la medición de distancia, es decir, hasta que actúa la señal Fin.

- *Listo:* Esta señal, también conectada a un LED, se enciende cuando se ha finalizado una medición y se ha refrescado el display, antes del inicio de la siguiente medición.

- *Fallo:* Esta señal hace que un LED se encienda únicamente cuando el sistema ha caído en un estado indeterminado del que no puede salir.

Las señales de entrada y salida que sirven de comunicación con la placa auxiliar se observan las siguientes:

- *Pulso:* Esta señal será un simple pulso que actúa durante 4 milisegundos cada vez que comienza una medición y se dirige hacia el driver del emisor en la placa auxiliar para la emisión de la onda de ultrasonidos.

- *Fin:* Dicha señal proviene del acondicionador del receptor en la placa auxiliar del medidor y supone la llegada del eco de la onda de ultrasonidos al receptor.

Según lo mencionado, en conjunto, el circuito codificado en VHDL y llamado Medidor deberá tener las siguientes funcionalidades:

- El circuito permanecerá con el display de 4 dígitos apagado y el LED de Apagado activo, sin realizar ninguna acción, siempre que el interruptor On_nOff esté en reposo. Cuando dicho interruptor sea accionado comenzará la actividad normal del medidor y se apagará el LED de Apagado.

- El circuito realizará medidas únicamente cuando, a parte de estar accionado el Interruptor On_nOff, se accione el interruptor de Inicio.

- Una medida será calculada a través del producto de la mitad de la velocidad del sonido con el tiempo de ida y vuelta de la onda de ultrasonidos. Este es el tiempo que transcurre desde que se da el flanco positivo en la señal llamada Pulso, hasta que se recibe la señal Fin de la placa auxiliar. El flanco positivo de la señal Pulso se dará cuando el display de 4 dígitos sea refrescado un número mínimo de veces tras la última medición y el Interruptor de Inicio sea accionado. En este periodo de tiempo permanecerá activo el LED Contando.

- El resultado de cada medida será pasado a números BCD y estos a su vez a código de 7 segmentos con punto para ser representados en el display de 4 dígitos, esto es el vector SieteSeg 7 a 0. El resultado tendrá precisión de Centímetros (mostrando cm) o de metros (mostrando M), según la medición. De tal forma habrá cuatro códigos en 7 segmentos, uno para cada dígito. Estos serán multiplexados temporalmente en el display mediante las señales AN 3 a 0 conectadas a los ánodos comunes de los dígitos (según lo explicado en el capítulo anterior).

- Dicho display será refrescado un número mínimo de veces tras cada medida para que la lectura sea estable y apreciable. El refresco mínimo se establece de tal modo que se realicen unas tres medidas cada segundo.

- El LED Listo permanecerá encendido cuando el Interruptor de Inicio no esté accionado o el display no haya finalizado su refresco mínimo.

- El Pulsador de Rearme reiniciará todos los registros, borrando el display y comenzando desde el primer estado del medidor como si este acabara de ser encendido con el Interruptor On_nOff.

- El LED de Fallo se encenderá muy raramente. Este indicador simbolizaría una pérdida de control del medidor. Para recobrar dicho control se deberá apagar y encender el medidor o bien pulsar el Rearme.

Con todo lo comentado en este epígrafe es fácil imaginarse que cada señal del circuito sintetizado tendrá una asignación determinada en el encapsulado de la FPGA. Dicha asignación será inequívoca para que cada señal sea actuada o actúe sobre el elemento de la placa de desarrollo Spartan III adecuado o sobre la placa auxiliar.

La asignación de señales del circuito Medidor al encapsulado de la FPGA se realiza mediante una aplicación incluida en el software de desarrollo que se comentará más adelante.

Teniendo en cuenta las funciones de las señales y las conexiones de los recursos de la placa Spartan III a la FPGA expuestas en el Capítulo Segundo, la asignación realizada será la que se muestra a continuación en la tabla 3.1:

Asignación de Pines

Nombre de la Señal	Dirección	Pin	Recurso
On_nOff	Entrada	F12	Interruptor junto PS/2
Inicio	Entrada	K13	Inter. junto Pulsadores
Fin	Entrada	D5	Pin5 Expansor A2
Rearme	Entrada	L14	Pulsador User Reset
Clk	Entrada	T9	Oscilador en Placa
Contando	Salida	P11	LED izquierdo
Listo	Salida	N12	LED medio-izquierdo
Fallo	Salida	N14	LED medio-derecho
Apagado	Salida	K12	LED derecho
Pulso	Salida	E6	Pin4 Expansor A2
AN(3)	Salida	E13	Ánodo Común 4º Dig.
AN(2)	Salida	F14	Ánodo Común 3º Dig.
AN(1)	Salida	G14	Ánodo Común 4º Dig.
AN(0)	Salida	D14	Ánodo Común 4º Dig.
SieteSeg(7)	Salida	E14	Segmento a
SieteSeg(6)	Salida	G13	Segmento b
SieteSeg(5)	Salida	N15	Segmento c
SieteSeg(4)	Salida	P15	Segmento d
SieteSeg(3)	Salida	R16	Segmento e
SieteSeg(2)	Salida	F13	Segmento f
SieteSeg(1)	Salida	N16	Segmento g
SieteSeg(0)	Salida	P16	Punto Decimal

Tabla 3.1: Asignación de pines del circuito Medidor a la FPGA

Todos estos son los requisitos que cumple el diseño del código VHDL sintetizado para obtener un circuito Medidor como el descrito, que cubra las especificaciones y que dote de la funcionalidad deseada al medidor de ultrasonidos desarrollado en este proyecto.

3.2. INTRODUCCIÓN AL LENGUAJE VHDL

El VHDL es un lenguaje de alto nivel orientado a la descripción de hardware para su síntesis en un componente electrónico programable y estándar; que es habitualmente una FPGA como la utilizada en este proyecto.

La palabra VHDL es el acrónimo de *VHSIC (Very High Speed Integrated Circuit) Hardware Description Languaje*, es decir, lenguaje de descripción de hardware para circuitos integrados de alta velocidad.

Esto quiere decir que con dicho lenguaje se pueden describir entidades hardware de carácter digital a partir de sentencias predeterminadas; pero siempre guardando las distancias con lo que se entiende por un lenguaje de programación de alto nivel. Nunca se debe olvidar que con VHDL se sintetizan circuitos que van a funcionar sobre un sistema físico como la FPGA en vez de ficheros ejecutables para un computador.

3.2.1. Marco Histórico

Durante los años setenta surgieron necesidades para el diseño de circuitos electrónicos. Una de estas necesidades, con las nuevas filosofías de diseño y sistemas EDA (*Electronic Design Automation*), era la de simular circuitos. De tal modo, era necesario un lenguaje para recrear entornos de simulación para circuitos, describiendo meramente su funcionamiento.

En ese momento, la única herramienta de descripción de Hardware eran las *Net List*, un código que especificaba las conexiones entre los componentes. Este código era de gran utilidad para trasladar esquemas a ficheros de intercambio en ASCII.

Con el fin de realizar la simulación funcional un lenguaje comenzó a ser desarrollado para el Departamento de Defensa de EEUU. durante los años ochenta. La investigación comenzó en 1981 como parte de un proyecto que pretendía tratar la crisis que estaban atravesando los circuitos en cuanto a su ciclo de vida.

En Julio de 1983, se contrató a un equipo de la empresa *Intermetrics (IBM y Texas Instrument)* para desarrollar las líneas principales sobre las que se basa el lenguaje. Ya en Agosto de 1985 se tenía la versión final de este lenguaje: la versión 7.2, en poder del gobierno Estadounidense. El equipo de desarrollo tomó como referencia el ya existente lenguaje ADA, con finalidades parecidas.

En 1986 se ceden los derechos del lenguaje a la IEEE *(Institute of Electrical and Electronics Engineers)*. De tal modo, en Diciembre de 1987 se publica el estándar *IEEE Std. 1076-1987* que como primera versión tiene el VHDL-87. En 1988 se convierte también en un estándar ANSI.

En Septiembre de 1993 se revisa todo el lenguaje y se obtiene una nueva versión, es la VHDL-93; que parte del estándar *IEEE Std. 1076-1993*.

Sin embargo, durante todo este proceso, el lenguaje fue diversificando sus posibilidades de utilización. Lo que originalmente solo serviría para simular circuitos ya existentes terminó siendo un lenguaje sin precedentes para describir nuevos circuitos y sus funciones. De hecho, aunque el lenguaje sea usado para describir Hardware, todavía contiene muchas de las funciones de simulación.

El propio Departamento de Defensa Estadounidense encargó una descripción comprensible de todos sus circuitos de aplicación específica *(ASIC)* de la serie *Military Standard 454*. Todo este trabajo fue realizado sobre el VHDL-87 en los años consecutivos a 1987.

De tal forma, se cubrieron más necesidades de las que inicialmente se pretendían solventar. Así se obtuvo un lenguaje que podría ser compartido por los integrantes de un grupo de trabajo o por varios grupos de trabajo, siempre común a todos. También se podrían reutilizar módulos ya desarrollados y probados anteriormente debido a la portabilidad que posee el VHDL frente a diversos fabricantes. Esto fue posible gracias al gran esfuerzo y trabajo de estandarización que supuso el VHDL.

Pero sin duda, lo que más ayudó a la imposición del lenguaje VHDL en el mercado fue su posibilidad de estructuración. El método *Top-Down* suponía la nueva forma de diseño electrónico; esto significaba partir de un modelo funcional a gran escala e ir adentrándose en el hasta obtener el sistema completo.

El diseño Top-Down tuvo una gran aceptación por sus resultados y el lenguaje VHDL permitiría seguir su filosofía a la perfección. De hecho, la mayor ventaja de este lenguaje consiste en sus niveles de abstracción que principalmente son dos: el estructural y el comportamental.

El nivel estructural permite la descripción de los circuitos mediante el conexionado literal de los componentes electrónicos como si fuera una primitiva Net List. Mientras tanto, el nivel comportamental, permite sintetizar circuitos mediante la descripción funcional de su comportamiento. La combinación de ambos niveles de abstracción y todos sus derivados supone el disponer de una potente herramienta de diseño y síntesis de circuitos que es el VHDL.

Así, el VHDL se perfila como un lenguaje de alto nivel y potencialidad para describir y simular circuitos sintetizables y programables sobre dispositivos como las FPGAs. Sin embargo, cabe mencionar al respecto que, a lo largo de la vida del VHDL, otros lenguajes de descripción hardware han convivido con el. El lenguaje que ha podido sobrevivir y establecerse junto con el VHDL es el *Verilog* de la firma *Cadence*.

Con todo lo mencionado, ya se pueden intuir las diferencias del VHDL frente a un lenguaje de programación Software de alto nivel, pero conviene reincidir en esto. No se puede programar en VHDL con una filosofía propia de un lenguaje convencional ya que el código resultante ha de sintetizar circuitos reales.

Un algoritmo convencional podría resultar algo imposible para una FPGA. Además, normalmente los eventos que suceden en una FPGA se procesan de forma concurrente y no secuencial, lo cual supone un gran cambio de concepción.

De tal forma, muchos de los circuitos que se plasman en un código VDHL son fijos y estrictos y sintetizan estructuras muy determinadas a las que pocos cambios se pueden añadir. Sin embargo, se adelanta que en este proyecto, respetando lo anteriormente mencionado y las limitaciones del VHDL; se ha seguido una filosofía de programación más abierta con algoritmos nuevos siempre siendo tolerados por los recursos de la FPGA al sintetizarse.

3.2.2. Sintaxis Básica en VHDL

Una vez introducidos los orígenes del lenguaje VHDL y sus características frente a lenguajes de programación de Software convencionales; se comentan a continuación los principios y estructuras básicas que deben constar en un programa VHDL.

En primer lugar, se deben enumerar y definir los principales objetos que se manejan en VHDL. Con estos objetos se podrán generar todo tipo de estructuras con el fin de sintetizar circuitos, siendo la base de unidades más complejas.

Existen tres clases principales de objetos, en función de su naturaleza y uso. Estas son las siguientes:

- *Constantes:* Es un objeto cuyo valor será invariable en el código en el que tenga vigencia. Su declaración se realizará como se observa a continuación: CONSTANT Nombre : Tipo := Valor. En este proyecto no es usa este tipo de objeto ya que lo sustituye otro llamado *GENERIC* que se explicará más adelante.

- *Señales:* Las señales son objetos con un significado físico, de carácter interno al circuito, que hacen referencia a los conductores del mismo. Las señales son muy comunes en cualquier estructura de

VHDL y se declaran mediante según lo siguiente: SIGNAL Nombre : Tipo. Estos objetos se asignan mediante el símbolo <= . Si se necesita obtener un conductor que una dos puertas lógicas se declarará una señal y posteriormente se utilizará en las puertas mencionadas.

- *Variables:* Las variables son elementos con significado abstracto en un circuito que pueden operarse numéricamente de forma mucho más sencilla que las señales. Este objeto solo tiene vigencia en el interior de la estructura de VHDL llamada proceso. Su peculiaridad es que se actualiza inmediatamente en estas estructuras, a diferencia de las señales. Se declaran como se muestra seguidamente: VARIABLE Nombre : Tipo. Estos objetos se asignan mediante el símbolo := .

En la anterior definición de los objetos que posee el lenguaje VHDL se ha hablado de su declaración. Además se le debe asignar un tipo a cada objeto. Estos tipos se pueden definir mediante la palabra *TYPE* o usar los predeterminados que aporta el lenguaje VHDL y las librerías declaradas.

En este proyecto se usan tipos estándar que se obtienen mediante las librerías que se llaman en cada código. Algunos de estos tipos son los siguientes:

- *STD_LOGIC:* Es el tipo más habitual que se usará para las señales de entrada y salida. Este tipo es de porte binario aunque contiene también otro tipo de estados que evitan ciertas situaciones forzadas que podrían dañar el circuito por una mala programación. Este tipo se encuentra en el paquete *std_logic_1164* de la librería IEEE.

- *STD_LOGIC_VECTOR:* Este tipo no es más que una agrupación del anterior de forma que se permiten estructuras de tipo vectorial para obtener líneas de datos semejantes a buses, etc. Para indicar el tamaño se debe adjuntar lo siguiente en su declaración: ((NumBits – 1) DOWNTO 0). Siendo NumBits el número de bits del vector y respetándose siempre el orden de mayor a menor en las asignaciones. Este tipo se encuentra en el mismo paquete que el anterior. Para conseguir una línea de siete conductores se declarará: SIGNAL Ejemplo : std_logic_vector (6 DOWNTO 0). Al comentar el código en esta memoria esta señal se hablará de las posiciones 7 a 1, aunque en el código VHDL aparece como en el ejemplo.

- *UNSIGNED:* Este tipo suele acompañar a la declaración de variables. Dicho tipo define un número siempre sin signo tratado de forma binaria de una dimensión especificada de idéntica forma a los vectores STD_LOGIC. Este tipo se encuentra en el paquete *std_logic_arith* de la librería IEEE.

- *POSITIVE:* Este tipo se ha utilizado para la descripción de datos genéricos. De tal forma se especifica que es un número entero y positivo.

- *INTEGER:* Este tipo siempre acompaña a variables. Como su nombre indica, define un número entero del cual se debe especificar su rango como se muestra a continuación: RANGE NumMin TO NumMax. Donde NumMin es el límite inferior y NumMax el superior.

Una vez expuestos los objetos y los tipos de objetos que se usan en el código de este proyecto y con los que cuenta el VHDL, se procede a describir los principales elementos para la descripción y síntesis de circuitos en este lenguaje.

Antes de pasar a las siguientes explicaciones sobre cada elemento en un código de VHDL se incluirán ejemplos extraídos de la herramienta de desarrollo que se ha utilizado en este proyecto y que se describirá en el siguiente epígrafe.

En cuanto dichos ejemplos se debe destacar que, el formato seguido corresponde al mismo usado durante el desarrollo del código VHDL sintetizado para el sistema desarrollado. Las palabras reservadas del lenguaje se escribirán en mayúsculas y en azul y los comentarios en verde.

Las tabulaciones y líneas espaciadoras son muy importantes para la organización del código y, en el que se ha desarrollado para este proyecto, se ha seguido un estricto orden sobre dichos elementos de estructurado. Sin más se expone la estructura rígida para generar un circuito en lenguaje VHDL.

Los elementos que se describirán a continuación son: el conjunto de librerías, la entidad, la arquitectura, el componente y el proceso. Estos elementos aparecerán en el interior de cada fichero *.vhd*. Algunos de ellos son imprescindibles y otros opcionales y ayudan a estructurar el diseño o conseguir ciertas funcionalidades características. La siguiente figura 3.2 es un esquema que refleja la relación entre los elementos que se describen en adelante:

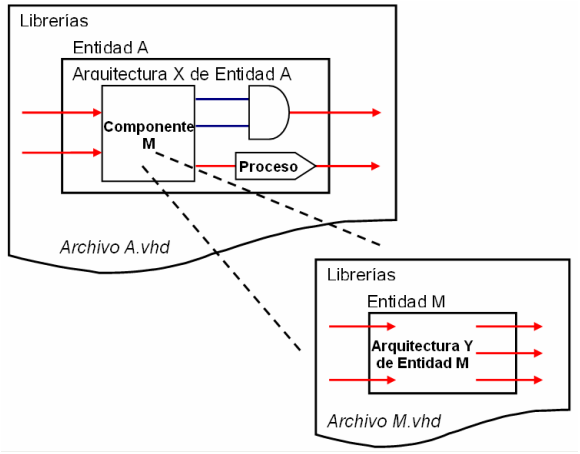


Figura 3.2: Estructura de los elementos VHDL

Lo primero que debe aparecer en un código VHDL es la declaración de las librerías y paquetes, soportadas por el lenguaje, que se van a utilizar en el código a sintetizar. Para la declaración de estas librerías y paquetes se escribirá algo semejante al siguiente código:

```
1  LIBRARY IEEE;                              --Librerias
2     USE ieee.std_logic_1164.all;
3     USE ieee.std_logic_arith.all;
4     USE ieee.std_logic_unsigned.all;
5
```

En este código, lo primero que debe aparecer es la palabra reservada *LIBRARY* junto con la librería de la cual se extraerán los paquetes; en este caso la *IEEE*. De esta librería se extraen diversos paquetes mediante la palabra *USE*.

El comienzo de cada paquete hace referencia a la librería de la que procede y el final hace referencia a que se incluyen todos los recursos del paquete. Entre los puntos se encuentra el nombre del paquete.

Ya desde el principio se pueden observar tres rasgos muy importantes del lenguaje: la sensibilidad del VHDL al carácter punto y coma; en vez del carácter retorno de carro ↵, la posibilidad de introducir comentarios mediante dos guiones --, y, reincidiendo, la importancia de utilizar tabulaciones para organizar la estructura del programa.

Como se observa en el código expuesto, existen distintos tipos de paquetes dentro de la librería IEEE. En este proyecto se utilizan tres de ellos que se describen a continuación:

- *std_logic_1164:* Este paquete suele estar presente en todo código VHDL y es el que contiene los tipos de señal que se usan para entradas y salidas así como señales internas.

- *std_logic_unsigned:* Este paquete permite realizar conversiones entre diversos tipos de variables y señales.

- *std_logic_arith:* Este paquete permite el uso de operadores matemáticos y relacionales así como ciertas conversiones de tipos de variables y señales.

Tras la declaración de librerías y paquetes que se utilizarán en el programa VHDL, lo siguiente que se debe introducir es la descripción del encapsulado del circuito. Esto permite la percepción del circuito a priori como una caja negra donde sólo se aprecian entradas y salidas del mismo; teniendo estructurado el diseño por exigencias del lenguaje.

El mencionado encapsulado del circuito es la llamada entidad del programa. Esta entidad se describe genéricamente el código que se muestra a continuación:

```
6   ENTITY Nombre IS                          --Nombre del Circuito
7      GENERIC (nombre_1 : tipo := valor;     --Datos Genéricos / Constantes
8               nombre_2 : tipo := valor;
9
10              nombre_n : tipo := valor);
11     PORT (nombre_1 : dirección tipo;       --Señales E/S del Circuito
12           nombre_2 : dirección tipo;
13
14           nombre_n : dirección tipo);
15  END Nombre;                               --Fin de la Descripción
```

Según se observa, para describir una entidad lo primero es darle un nombre, lo cual se realiza escribiendo el mismo entre las palabras reservadas *ENTITY* e *IS*.

Seguidamente se pueden o no definir datos genéricos mediante la palabra *GENERIC*. Gracias a estos datos se podrán fijar constantes que permanecerán invariables en el diseño y que, si es necesario reutilizar el circuito, serán fácilmente modificadas. Cada línea acabará con el carácter ; excepto la última, donde dicho carácter se escribirá tras el paréntesis de cierre que marca el final de declaración de datos genéricos.

Con lo que siempre debe contar una entidad es con la definición de entradas y salidas, es decir, los puertos. Esto se realiza mediante la palabra

PORT. Cuando son del mismo tipo y dirección los nombres se pueden agrupar siendo separados por comas en la misma línea. La forma desarrollar y finalizar la asignación es semejante a la de los datos genéricos.

Por último se debe cerrar la definición de la entidad con la palabra *END* y el nombre de la entidad. Siempre se acabará con el carácter de fin de línea, es decir, el punto y coma.

A continuación, se describirá el contenido la entidad, es decir, la arquitectura de la misma. Este es el punto en el que realmente se definen los circuitos según los procedimientos que se explicarán más adelante y donde se usan las entradas, salidas y datos genéricos que se declaran en la entidad.

Por lo tanto, la arquitectura es el fragmento de código donde se describen todas las conexiones y funciones del circuito que se encuentra dentro del encapsulado que define la entidad. Dicho elemento se declara genéricamente según lo siguiente:

```
17  ARCHITECTURE NombreArq OF NombreEnt IS --Nombre de la Arquitectura
18                                         --Parte Declarativa
19     COMPONENT NombreComp_n                 --Declaración Componentes
20        GENERIC (Datos Genéricos);
21        PORT (Entradas y Salidas);
22     END COMPONENT;
23
24     SIGNAL nombre_1 : tipo;                --Declaración Señales
25     SIGNAL nombre_2 : tipo;
26
27     SIGNAL nombre_n : tipo;
28
29     BEGIN                                  --Parte Descriptiva
30                                               --Utilización de Componente
31        Componente : NombreComp_n PORT MAP (Asignación Entradas y Salidas);
32                                               --Asignación Directa
33        nombre_1 <= nombre_2 AND nombre_n;
34
35  END NombreArq;                             --Fin de la Arquitectura
```

Lo primero que se aprecia es la forma de nombrar la arquitectura que es, a la vez, la de comenzar su descripción. Esto se realiza mediante la palabra reservada *ARCHITECTURE* seguida del nombre de la arquitectura y la palabra *OF* con el nombre de la entidad a la que pertenece y finalizando con la palabra *IS*.

Así se entra en el espacio de la parte declarativa de la arquitectura. Aquí se pueden definir diversos elementos a usar donde el desarrollo de la arquitectura. Un elemento al que se hará mención más adelante es el *COMPONENT*, que facilita mucho la filosofía Top-Down.

Otro elemento muy común son las señales internas que se usarán para tratamientos intermedios entre entradas y salidas. Una señal se declara mediante la palabra *SIGNAL* seguida del nombre de la misma y el carácter dos puntos : finalizando con el tipo de objeto y el carácter de fin de línea.

Posteriormente a la declaración de elementos de la arquitectura, la parte descriptiva comenzará mediante la palabra *BEGIN*. A partir de este punto será donde se realice realmente la descripción como tal de los circuitos de la arquitectura y por ende de la entidad.

Por lo tanto, en este espacio se podrán usar los componentes o señales anteriormente declarados. Se puede observar también que para realizar la asignación sobre señales en esta parte descriptiva se usa la combinación de caracteres <=, apuntando el primer símbolo hacia el resultado, como se mencionó anteriormente. También cabe mencionar como nombrando la palabra *AND*, en este caso, se hace referencia al operador lógico del mismo nombre.

Por último y para finalizar la arquitectura, se escribe la palabra *END* seguida del nombre de la arquitectura y del carácter ; de fin de línea.

El elemento *COMPONENT* antes mencionado no es más que un subcódigo de VHDL que se reutiliza en una arquitectura superior para poseer finalmente una funcionalidad mayor. Este archivo de subcódigo debe residir en el mismo directorio donde se encuentre el archivo que lo llama. El ejemplo más común para explicar este elemento es la síntesis de un sumador completo a partir de dos semisumadores que son los componentes.

De tal modo, para utilizar un componente se debe declarar en la arquitectura. Esta declaración no es más que la entidad del componente a utilizar. Sin embargo se debe cambiar la palabra *ENTITY* por *COMPONENT*, eliminar la palabra *IS* y finalizar con *END COMPONENT*; . De esta forma, la arquitectura donde se declara tiene en cuenta los puertos de entradas y salidas que posee el componente.

Para utilizar este componente en la parte descriptiva de la arquitectura se debe etiquetar al mismo mediante un nombre al que hacer referencia seguido del carácter : . Tras esta etiqueta se escribe el nombre de la entidad tal y como se declaró anteriormente seguido de las palabras *PORT MAP*. Es en ese momento cuando se introducen entre paréntesis todas las señales de entrada y salida que requiera el componente, siempre separadas por el carácter , . Finalmente se terminará con el carácter de fin de línea.

Hasta este punto se han comentado las estructuras principales del lenguaje VHDL y los elementos las dotan de funcionalidad. Se sabe como organizar correctamente un programa y como se puede seguir la metodología de diseño Top-Down.

Sin embargo, para poder implementar circuitos complejos todavía se ha de profundizar en los dos tipos principales de algoritmos sintetizables y las sentencias que los caracterizan.

Estos dos tipos de algoritmos son los concurrentes y los secuenciales. Por defecto en el lenguaje VHDL se toma el código como concurrente ya que el producto final es la síntesis de un circuito digital que es independiente del orden en el que se escriban.

Es por esto que para conseguir que una sentencia se ejecute secuencialmente se debe recurrir a una estructura anteriormente mencionada llamada proceso. Esta estructura se incluye siempre dentro de la parte descriptiva de la arquitectura según el código mostrado a continuación:

```
29    BEGIN                                --Parte Descriptiva
30
31       Componente : NombreComp PORT MAP (Asignación Entradas y Salidas);
32
33       nombre_a <= nombre_b AND nombre_c;
34
35       PROCESS (Lista de Sensibilidad)   --Declaracion de Proceso
36
37          VARIABLE nombre_1 : Tipo;      --Parte Declarativa
38          VARIABLE nombre_2 : Tipo;
39
40          VARIABLE nombre_n : Tipo;
41
42          BEGIN                          --Parte Descriptiva Secuencial
43                                            --Asignación Variables
44             nombre_1 := nombre_2 OR nombre_n;
45                                            --Desarrollo del Proceso
46
47          END PROCESS;                   --Fin del Proceso
```

En este fragmento de código se observa como esta estructura llamada proceso debe ser declarada siempre en el desarrollo de la parte descriptiva de una arquitectura. Esta declaración se realizará mediante la palabra *PROCESS* seguida de un paréntesis en cuyo interior se enumerarán todas las señales pertinentes separadas por el carácter coma , .

A todas las señales incluidas entre los paréntesis se le llama lista de sensibilidad. Esto significa que cuando cambie alguna de estas señales y solo en ese momento será cuando se ejecute el contenido del proceso. Si no existiera esta lista el proceso se ejecutaría continuamente.

Lo siguiente que se aprecia es una parte declarativa dentro de este proceso donde se podrán declarar típicamente variables tal y como se observa en la figura y se mencionó anteriormente.

Tras la parte declarativa, comienza la parte descriptiva y puramente secuencial mediante la palabra *BEGIN*. A partir de este momento las sentencias se ejecutarán siempre de modo secuencial y teniendo en cuenta el orden de las mismas.

En esta parte se puede observar un ejemplo de una asignación de variables mediante el símbolo := . Es importante hacer referencia a que esa asignación se produce en el instante en el que se ejecuta la línea de la secuencia donde se encuentra la misma. Sin embargo, si la asignación fuese de señales, esta no tendría efecto hasta finalizar la ejecución del proceso.

Después de todo lo que acontezca en el desarrollo de la parte descriptiva del proceso se debe finalizar el mismo mediante las palabras *END PROCESS* y el carácter de fin de línea ; .

Cabe mencionar que, los algoritmos que se pueden sintetizar circuitos semejantes de forma secuencial y concurrente pero con distintas sentencias. En este proyecto se utiliza siempre el tipo de algoritmo secuencial, observándose y explicándose sus secuencias durante el desarrollo de los siguientes epígrafes.

Con todo lo descrito en este epígrafe es posible comprender mejor la función del lenguaje VHDL en este proyecto así como el propio código que se ha desarrollado para su correcto funcionamiento.

De tal modo, todo lo aquí descrito se ha pasado por el filtro de lo meramente utilizado en este proyecto, es decir, que existen diversas formas de realizar un mismo paso en VHDL; pero la aquí descrita es la usada para el código del medidor.

3.3. ENTORNO DE DESARROLLO

Cuando se haya generado un código en VHDL válido, es necesario disponer de un software específico para sintetizarlo y volcarlo sobre la FPGA. Este software es propio de cada fabricante e incluso, de cada familia de FPGAs; pero siempre debe ser totalmente compatible con VHDL.

Este Software dota, junto a la placa de desarrollo, a la FPGA de una gran flexibilidad a la hora de realizar un desarrollo y su prototipo. Un Software para este tipo de aplicaciones permite las siguientes funciones:

- Edición de código VHDL para su creación en un entorno amigable que señaliza las palabras reservadas del lenguaje así como otros elementos y es sensible a las sangrías.

- Agrupación de diversos códigos y archivos VHDL según un modelo relacional en forma de árbol mediante elementos como los componentes.

- Edición y programación de FPGA a través de esquemáticos y símbolos dejando a un lado el lenguaje VHDL y conversión de código VHDL a símbolos.

- Edición y asignación de entradas y salidas del código VHDL de mayor nivel al pin deseado del encapsulado de la FPGA, siempre que la misma lo permita.

- Síntesis del código VHDL generado a circuitos en un código comprensible por la FPGA.

- Generación de archivos pertinentes para simulación por ordenador y simulación por ordenador de los códigos VHDL sintetizados para verificar su funcionamiento antes de la carga en la FPGA.

- Implementación final de código a cargar en la FPGA. En este paso se utilizan los productos de la asignación de pines y de la síntesis de código para dar lugar al producto final que solo servirá para cargarlo en una FPGA determinada.

- Generación del fichero final de programación, tanto en memoria volátil de FPGA como en memoria no volátil externa, y programación total de dichos elementos según se desee.

Con estas herramientas y funciones siempre se pretende conseguir una metodología de diseño eficaz y que permita verificar y depurar todo tipo de errores y fallos. Esta filosofía impone una constante revisión del diseño hasta obtener un producto totalmente fiable y con un coste mínimo de prototipado. El flujo de diseño concurrente que se ha de seguir para llevar a cabo dichos métodos se expone en la figura 3.3:

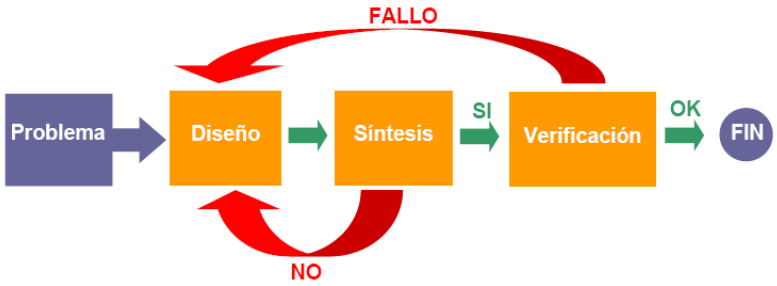


Figura 3.3: Flujo de diseño concurrente

Cabe mencionar que, las funciones utilizadas para la elaboración del presente proyecto, de son: la edición y agrupación de código, la asignación y edición de pines de la FPGA, y todo lo referente a la generación de todos los ficheros pertinentes para la programación de la FPGA.

De entre las funciones que se han expuesto en la anterior enumeración; no se utiliza la simulación por computador ya que, a pesar de ser una herramienta muy potente, el hecho de disponer de una placa de desarrollo ha facilitado todos los test. Esta placa ha permitido que cualquier prueba y simulación pudiera ser realizada sobre el dispositivo real. Estas pruebas son incluso más fiables que las de la simulación por computador, pese a su complejidad y desarrollo hoy día.

Esto ha sido posible debido a la naturaleza, función y tamaño de los circuitos diseñados. Es probable que para circuitos con funciones menos visibles o con demasiadas entradas y salidas no sea viable realizar las simulaciones como se han llevado a cabo en este proyecto. Sin embargo siempre se debe contar con una etapa de verificación y pruebas para seguir el esquema de la figura 3.3 y sacar adelante un proyecto con éxito y eficacia.

Descritas todas las características y funciones que se esperan de un Software para el desarrollo de un circuito para FPGA y comentadas las funciones que se utilizan y son necesarias para el desarrollo del medidor de ultrasonidos; se continuará introduciendo el Software específico utilizado en este proyecto.

La herramienta Software utilizada para el correcto desarrollo de este proyecto se denomina comercialmente *ISE* WebPACK™. Este software no se facilita con la placa de desarrollo Spartan III. Sin embargo no resulta muy complicado conseguirlo a través de Internet de forma gratuita en la página web del fabricante de la FPGA: *www.xilinx.com*.

Este software cumple con todos los requerimientos anteriormente expuestos. De hecho, los pasos a seguir que se deben realizar para conseguir la programación de un código VHDL en la FPGA se corresponden en su totalidad con lo anteriormente comentado.

Estos pasos, específicos para el Software ISE *WebPACK*™, se muestran en la siguiente figura 3.4:

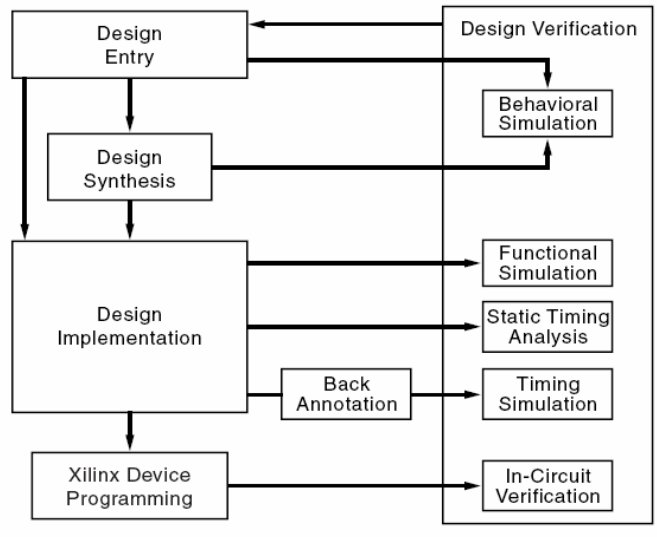


Figura 3.4: Diagrama de procesos de ISE *WebPACK*™

Como se aprecia en la figura, la columna de la izquierda son todos los procesos que conlleva la programación de un diseño en VHDL en la FPGA y la columna de la derecha muestra todo lo referente a la verificación del diseño. Anteriormente ya fue comentado que en este proyecto, gracias a la placa de desarrollo, la verificación se realizo en la propia FPGA, adaptando las entradas y salidas.

La descarga e instalación del Software ISE *WebPACK*™ da como resultado la disponibilidad de un programa principal llamado *Project Navigator* y varias aplicaciones independientes que pueden ser llamadas desde el.

De tal modo, para realizar cualquier acción sobre el diseño o la FPGA se cargará el programa Project Navigator desde el cual, organizadamente, se irán llamando a las aplicaciones pertinentes. Por lo tanto, todo lo que se describe a continuación se realiza desde el programa Project Navigator.

3.3.1. Generación de un Proyecto

Para comenzar un diseño, lo primero que se debe hacer es generar un proyecto. Este programa parte de que dentro de este proyecto habrá diversos códigos VHDL o esquemáticos y, durante la generación del proyecto, ayuda a la creación de dichos archivos paralelamente. Para ello se utilizará el programa Project Navigator.

Una vez abierto el Project Navigator, la ruta que se debe seguir para generar un proyecto es *File → New Project*. Al ejecutar esto, el programa muestra un cuadro de diálogo como el que se muestra a continuación:

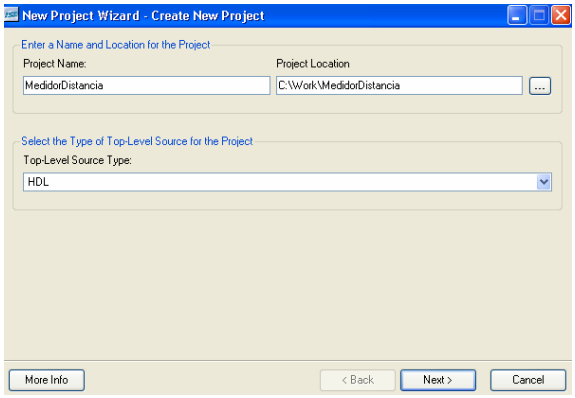


Figura 3.5: Ventana "Create New Project" de ISE *WebPACK*™

En la figura 3.5 se observa como el cuadro de diálogo pide un nombre y carpeta donde guardar el proyecto que se va a generar. También se puede seleccionar de qué tipo será el módulo de más alto nivel. En este proyecto se trabajará siempre con VHDL en vez de Verilog o esquemáticos, por lo tanto esta opción será la de HDL como se aprecia en la figura.

Al proseguir en la generación del proyecto, pulsando sobre el botón *Next>*, aparece un cuadro de diálogo donde se debe especificar la FPGA sobre la que se desarrolla el proyecto:

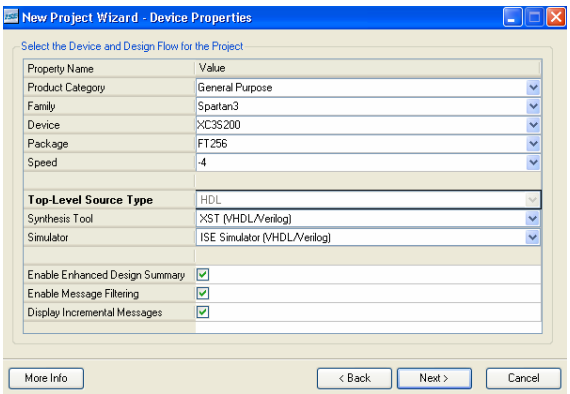


Figura 3.6: Ventana "Device Properties" de ISE *WebPACK*™

Lo siguiente que se pide es una lista con los nuevos ficheros VHDL a insertar en el proyecto. Aquí se pueden especificar el nombre de entidad y arquitectura y las entradas y salidas para facilitar la tarea de diseño generando automáticamente el código VHDL resultante.

Sin embargo, suponiendo que los archivos VHDL han sido codificados en un editor cualquiera y se han guardado con extensión *.vhd*, también se podrán agregar al diseño tal y como se muestra en el cuadro de diálogo que sigue al anteriormente comentado. Esto se observa en la siguiente figura:

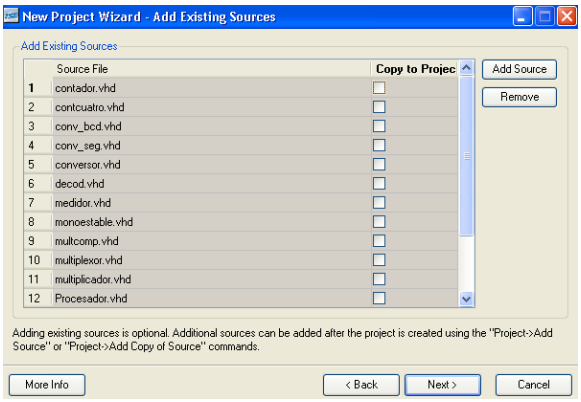


Figura 3.7: Ventana "Add Existing Sources" de ISE *WebPACK*™

En esta figura 3.7 se muestra como con el botón *Add Source...* se podrán incluir archivos de código VHDL ya existentes explorando en el disco duro del PC. La casilla *Copy to Project* copiará el archivo desde su localización a la carpeta del proyecto, lo cual es muy cómodo.

Por último, al pulsar sobre el botón *Next >*, aparecerá el cuadro de diálogo final donde se muestra un resumen con todas las opciones y características seleccionadas para el proyecto. Para concluir este proceso se pulsará sobre el botón *Finish*. En ese momento se muestra un nuevo cuadro de diálogo para clasificar los archivos como bancos de pruebas o circuitos a sintetizar. La opción predeterminada es correcta, por lo tanto se dará al botón *OK*.

Una vez creado y procesado el proyecto con todos los archivos .vhd ya codificados en su interior, el programa Project Navigator muestra la siguiente imagen (Figura 3.8):

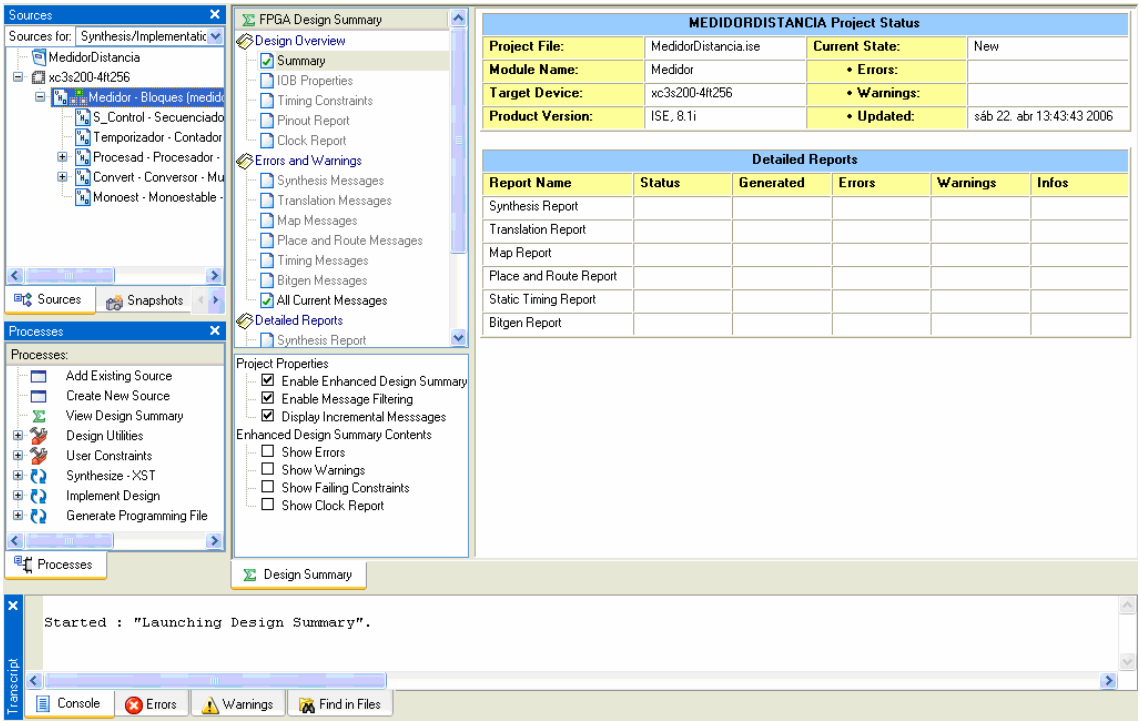


Figura 3.8: Entorno de trabajo de ISE *WebPACK*™

Como se observa en la figura 3.8, en la parte superior se encuentran los menús habituales y justo debajo la barra de herramientas correspondientes para el acceso por iconos a diversas funciones. También se observa como el espacio de trabajo se divide en cuatro partes:

- En la parte superior izquierda se sitúa el cuadro de archivos fuente. Aquí se podrán encontrar todos los archivos que están anexos al proyecto en forma de árbol.

- En la parte media izquierda se observa el cuadro de procesos. En este espacio se dispondrá de todos los procesos posibles para el archivo seleccionado en el cuadro de archivos fuente.

- En la parte derecha se encuentra el cuadro de informes y edición. Aquí irán apareciendo los resúmenes de los procesos realizados y se podrán editar los archivos de texto pinchando sobre ellos en el cuadro de archivos fuente. También se situarán en este espacio las ventanas de conexión mediante Internet al soporte de Xilinx.

- En la parte inferior se mostrarán todas las acciones que vaya realizando el programa en tiempo real, acorde con los procesos o

funciones seleccionados. También se agruparán todos los errores y avisos por separado y, pinchando sobre su vínculo se podrá visitar automáticamente la página de soporte de Xilinx que da solución al mismo.

3.3.2. Síntesis

Aquí describe cómo realizar la síntesis del circuito codificado en VHDL y una vez generado el proyecto.

Según lo comentado en el epígrafe anterior y lo mostrado en la figura 3.8, existe un espacio donde se muestran los procesos disponibles para cada archivo llamado cuadro de procesos.

Para que la opción de síntesis llamada *Synthesize – XST* se encuentre disponible; en el cuadro de archivos fuente debe estar seleccionado el archivo de mayor nivel, es decir, el que sintetice el circuito total.

Como ya se mencionó anteriormente, el ISE *WebPACK*™, incluye el Project Navigator que se está comentando y aplicaciones son utilizadas por él. Una de estas aplicaciones es la de síntesis y se llama XST *(Xilinx Synthesize Technologies)*; de ahí el nombre del proceso. Esta aplicación genera un fichero de extensión *.ngc* a partir del fichero *.prj* donde residen todos los archivos presentes en el proyecto activo.

Una vez se tiene seleccionado el archivo .vhd de mayor jerarquía, se pulsará con un doble clic sobre el icono del proceso de síntesis en el cuadro de procesos. Esto hará que automáticamente se comience a sintetizar todo el código.

Mientras se está ejecutando el proceso de síntesis aparecen en el cuadro inferior todos los mensajes relativos a la síntesis. En este periodo de tiempo una esfera animada gira junto al nombre del proceso en el cuadro de procesos.

Al finalizar se observa como en el cuadro de procesos se muestra un tic de color verde junto al proceso de síntesis que simboliza que esta ha sido llevada a cabo correctamente.

También se puede observar que en el cuadro de informes aparece otro tic verde sobre el folio de *Synthesis Messages* o *Synthesis Report*, lo cual significa que este informe se encuentra disponible para observar todo el

sumario del desarrollo del proceso. Se observarán así los errores y avisos que ha supuesto la síntesis.

Todo esto se observa en la siguiente figura 3.9:

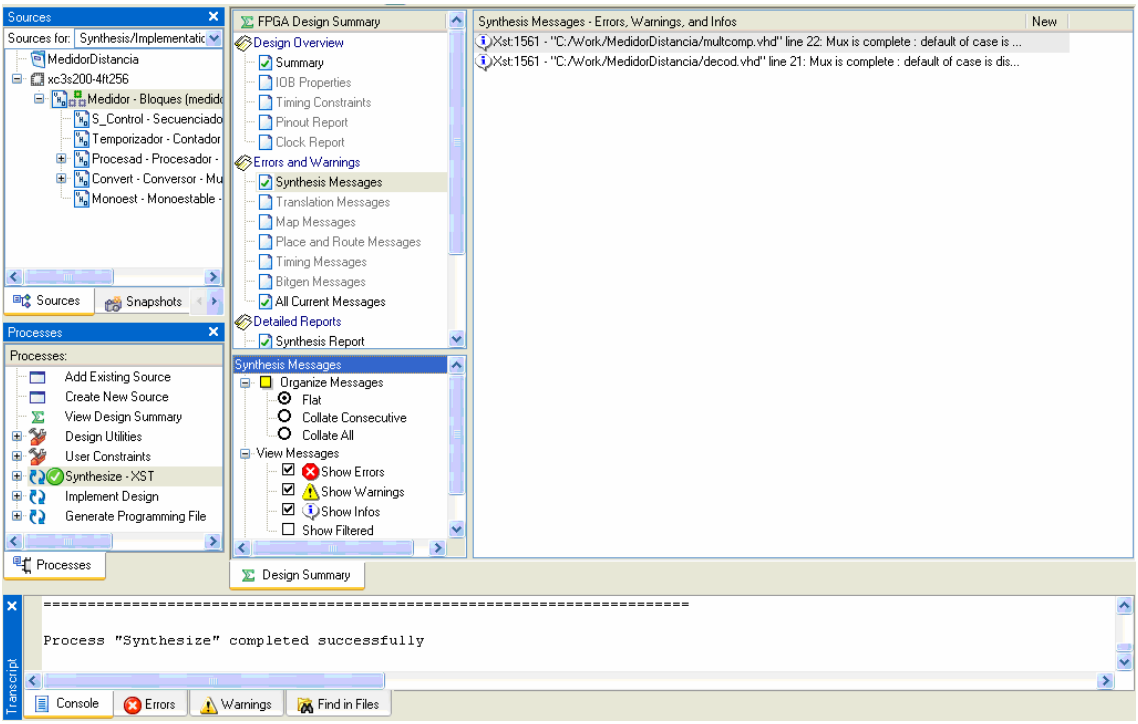


Figura 3.9: Finalización de proceso de síntesis en ISE *WebPACK*™

Si en vez del tic verde, junto al proceso de síntesis apareciese un aspa roja, esto significaría que existen errores críticos que impiden continuar con la síntesis y que esta no se ha llevado a cabo.

Además, también existe la posibilidad de que junto al proceso de síntesis aparezca una admiración ámbar. Esto significa que el proceso de síntesis se ha concluido con algunos errores o avisos de tipo leve pero que pueden afectar al funcionamiento del circuito final. En ese caso, lo más conveniente es visitar la página web de soporte de Xilinx para saber lo que puede afectar el aviso.

Si se pulsa con el botón derecho del ratón sobre el proceso de síntesis se abre un desplegable con las funciones que ese proceso puede realizar (se puede recargar, interrumpir, etc.) y con las propiedades. Pulsando sobre esta función aparece un cuadro de diálogo para cambiar las propiedades de síntesis a gusto del usuario.

Cambiar estas opciones tiene efectos muy activos sobre la síntesis de un circuito. Estos cambios pueden generar dos circuitos dentro de la FPGA muy distintos partiendo del mismo código VHDL. En general se usarán las

propiedades determinadas salvo que el propio programa especifique que un circuito puede ser más eficientemente sintetizado con otras opciones.

Para mayor detalle y referencias al respecto se puede consultar el DVD que acompaña a esta memoria, en el que se adjunta documentación de este Software.

3.3.3. Asignación de Pines FPGA

Este proceso se realiza mediante la manipulación del usuario de un programa del paquete de ISE *WebPACK*™ llamado *PACE*. De tal forma y conociendo la numeración de pines de la FPGA que se comentó en el Capítulo Segundo se puede utilizar el encapsulado del circuito de forma muy flexible.

Cabe comentar que gracias a esta propiedad, el conocimiento de la placa de desarrollo Spartan III y la conexión de los elementos de la placa a la FPGA; se pueden llegar a utilizar correctamente todos los recursos de esta placa de desarrollo.

También es importante saber que sin esta asignación de pines no tiene sentido continuar con los procesos que terminan por obtener un fichero para volcar a la FPGA. Esto se debe a que este circuito no tendrá ningún tipo de funcionalidad sobre la placa de desarrollo si la asignación de pines no se hace a conciencia.

Para ejecutar el programa PACE se debe pulsar sobre el símbolo + que aparece al lado del icono del proceso *User Constraints* del cuadro de procesos. En ese momento aparecen nuevos procesos y uno de ellos es *Assign Package Pins*.

Al hacer doble clic sobre este proceso se pregunta si se desea crear un archivo de tipo .ucf para guardar la asignación de pines ya que no hay ninguno generado anteriormente. Si se responde que si, se abre el programa PACE y se comienza con la configuración de pines de la FPGA.

Este programa detecta automáticamente cuáles son los pines de entrada y salida que posee el circuito del diseño seleccionado en el cuadro de archivos fuente y se muestra en la pantalla el siguiente entorno de trabajo (figura 3.10):

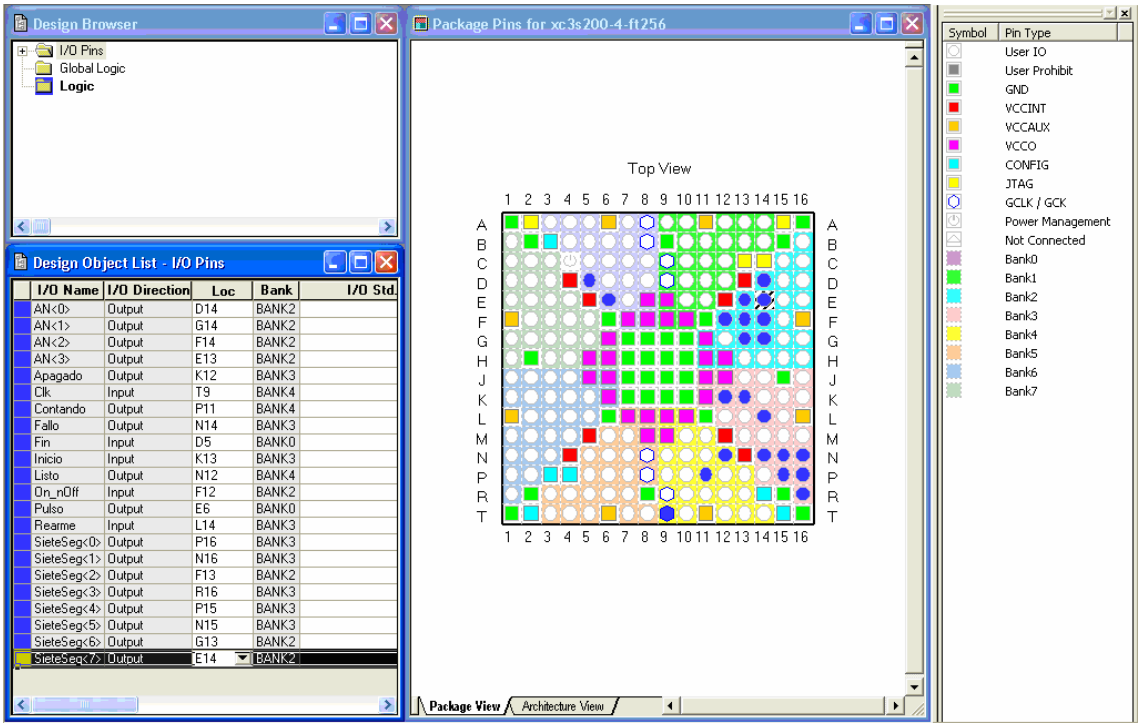


Figura 3.10: Proceso de asignación de pines en PACE

Como se aprecia en esta figura 3.10, la pantalla se divide en varios espacios de trabajo. El esquema de los pines de la FPGA y su leyenda son meramente informativos. Lo importante en este programa es la lista de señales del diseño que aparecen en la parte inferior izquierda de la pantalla.

En esta lista se muestran todas las señales que se han especificado en la entidad del archivo fuente seleccionado en el cuadro correspondiente del Project Navigator. Una de las columnas de esta lista se titula Loc.

Pues bien, el proceso de asignación de pines de la FPGA es tan sencillo como describir en esta columna el pin de la FPGA (la letra de fila y número de columna) que le corresponde a cada señal del diseño realizado.

Al mismo tiempo que se asignan estas señales, en el gráfico de los pines de la FPGA se colorean en azul los pines usados. Quedando al final un dibujo como el que se muestra en la figura.

La asignación de pines que aparece en la figura es la misma que se ha llevado a cabo en este proyecto. Sobre esta asignación se hablará más adelante cuando, a parte de conocer la placa de desarrollo y elementos

puramente hardware, se conozcan también los requerimientos de la FPGA y el código VHDL generado.

Al finalizar la asignación se debe grabar para salvar todos los valores introducidos. Esto hace que en el cuadro de archivos fuente del Project Navigator aparezca instantáneamente un nuevo archivo del mismo nombre que el del circuito sobre el que se ha hecho la asignación pero con extensión *.ucf*.

En este momento ya se puede cerrar la aplicación PACE sin preocuparse por perder la asignación de pines realizada y sabiendo que dicha asignación será tenida en cuenta en las siguientes fases de generación del fichero para programar la FPGA. Además, se puede consultar la asignación en el folio *IOB Properties*.

3.3.4. Implementación del Diseño

Una vez se ha llevado a cabo la asignación de señales del circuito sintetizado a los pines de la FPGA se está en disposición de ejecutar un nuevo proceso del cuadro de procesos del Project Manager.

A este proceso se le llama implementación del diseño y tiene como fin la generación del archivo a partir del cual se genera la información a volcar sobre la FPGA. El proceso en sí se divide en tres partes, las cuales se describen a continuación por orden de ejecución:

- *Translate:* Es la llamada traducción, que implica la fusión de los archivos de síntesis con extensión *.ngc* y de asignación de pines con extensión *.ucf* en uno solo que unifica toda esta información; cuya extensión es *.ngd*. Este proceso se realiza mediante la aplicación *NGDBUILD*.

- *Map:* También conocido como mapeado y es el proceso que a través del fichero *.ngd* decide y aplica los requerimientos de los bloques de la FPGA que se van a utilizar para satisfacer las exigencias del diseño. Esto se guarda en varios ficheros de tipo *.pcf* y *_map.ncd* y el nombre de la aplicación es *MAP*, al igual que el proceso.

- *Place & Route:* Este proceso llamado emplazamiento y rutado es el que posiciona los bloques anteriormente designados en lugares físicos de la estructura de la FPGA y los conexiona para obtener la funcionalidad deseada. Se parte de los archivos de extensión *.ngd*, *.pcf* y *_map.ncd* para guardar el resultado en el archivo *.ncd* del mismo nombre que el archivo *.vhd* que del que parte la síntesis. Esta aplicación se llama *PAR*.

Haciendo doble clic sobre el proceso *Implement Design* en el cuadro de procesos se ejecutarán seguidamente estos tres subprocesos.

Al finalizar el proceso de implementación del diseño con éxito, ya ejecutadas las tres partes anteriormente descritas, se muestra un tic verde al lado del proceso; semejante al de la síntesis.

También se pueden dar los mismos casos que en la síntesis, es decir, que se muestre una admiración amarilla para errores no críticos y un aspa roja para errores que impiden la finalización del proceso.

Esto se observa claramente en la figura 3.11:

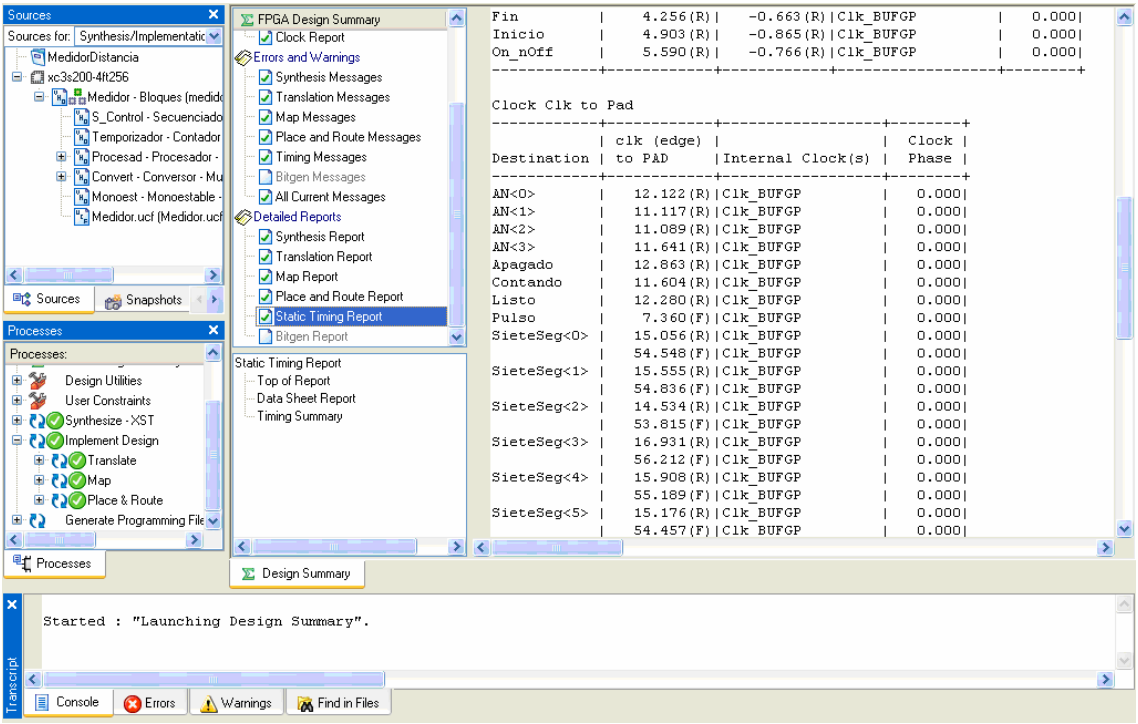


Figura 3.11: Finalización de proceso de implementación en ISE *WebPACK*™

En dicha figura se puede apreciar claramente el tic del proceso de implementación así como los de los tres subprocesos.

En cualquier caso y para más detalle, especialmente en el caso de errores, al igual que en la síntesis también en este proceso se generan resúmenes de errores y avisos y de desarrollo del proceso. En esta figura también se observan la presencia y disponibilidad de estos informes marcados con un tic verde sobre un folio.

Los errores y avisos se consultan pulsando sobre cada uno de los nombres de los subprocesos en el apartado *Errors And Warnings*, a la izquierda del cuadro de informes y edición. Los informes de desarrollo y ejecución de los subprocesos se encuentran debajo de los anteriores en el grupo llamado *Detailed Reports* y se muestran pulsando sobre ellos.

Un informe sintetizado adicional que se muestra al final de la implementación del diseño, fruto del estudio de la respuesta temporal del circuito diseñado e implementado físicamente; es el llamado Informe Estático de Sincronización, es decir, *Static Timing Report*. Tal y como se observa en la figura 3.9 este informe se encuentra abierto y en él se pueden consultar los retardos de las señales del diseño.

3.3.5. Programación de la FPGA.

Como se puede intuir, este es el proceso que culmina con el volcado del diseño del circuito a través de código VHDL en este caso a la FPGA.

Existen dos formas de realizar este volcado según se explicó en el Capítulo Segundo al comentar la estructura de la FPGA utilizada. De esta forma, se podrá programar dicha FPGA sobre un soporte volátil o de forma no volátil en la memoria Flash de la placa de desarrollo Spartan III. Ello conllevará dos procesos distintos.

Para ambos procesos siempre será necesaria la creación del archivo de programación a partir del archivo de extensión *.ncd* con la información final del diseño después de aplicarle todos los procesos. Para ello se puede hacer doble clic sobre el cuadro de procesos en *Generate Programming File* y este proceso llamado *BITGEN* (*Bitstream Generation*) se realizará automáticamente.

En este momento y al finalizar la generación del archivo programable de extensión *.bit*, se mostrará un tic verde semejante a los anteriormente comentados en los otros procesos. También se ofrecerá un nuevo informe llamado *Bitgen* así como el resumen de errores y avisos. Esto se muestra en la siguiente figura 3.12:

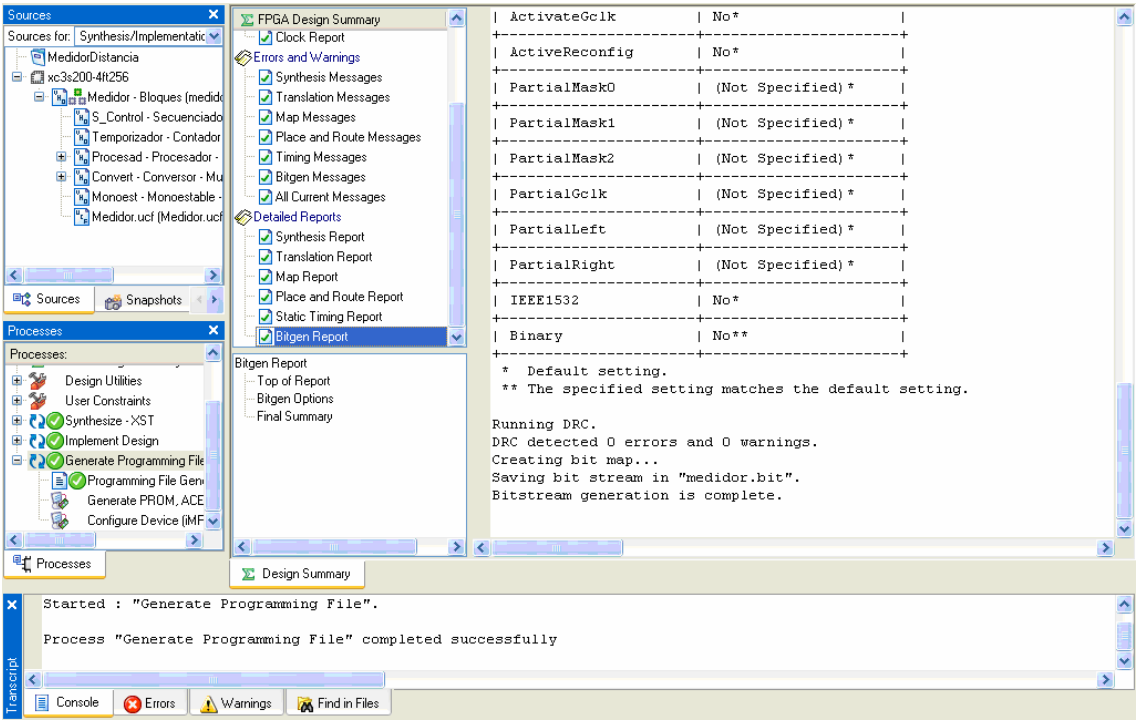


Figura 3.12: Generación de archivo programable en ISE *WebPACK*™

A parte de todo lo comentado, en esta figura se observa como el proceso *Generate Programming File* también cuenta con dos subprocesos.

Si se pretende realizar una programación en memoria Flash no volátil del diseño, habrá que generar un archivo de tipo *.mcs* mediante la aplicación *iMPACT* que se observa en la figura 3.12. Este programa se abrirá pulsando dos veces sobre él apareciendo el cuadro de diálogo de la figura 3.13:

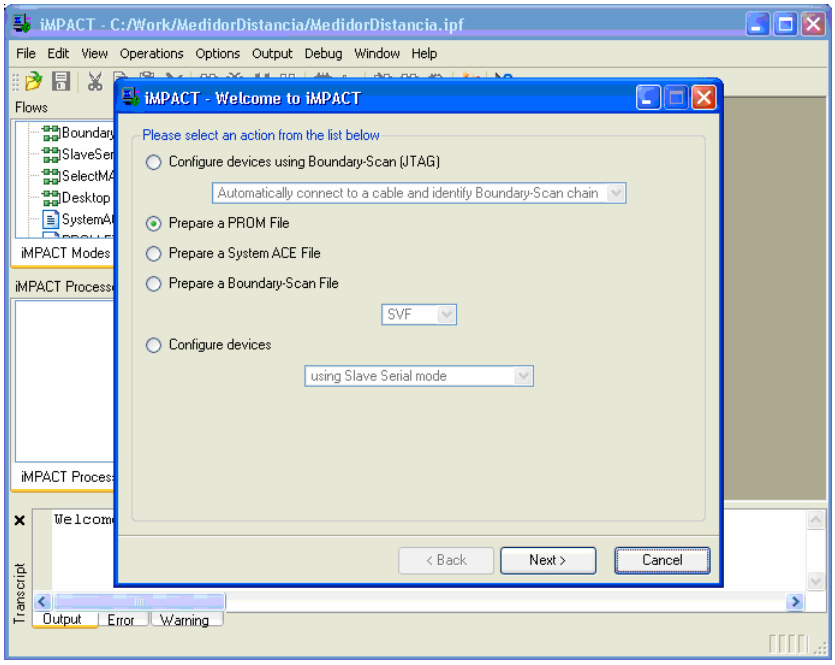


Figura 3.13: Ventana de inicio de iMPACT

Como se aprecia en esta figura, la opción que se debe marcar para obtener el fichero de programación de la FPGA es la de *Prepare a PROM File*. Al pulsar sobre el botón *Next*>, se muestra otro cuadro de diálogo como el que se muestra a continuación:

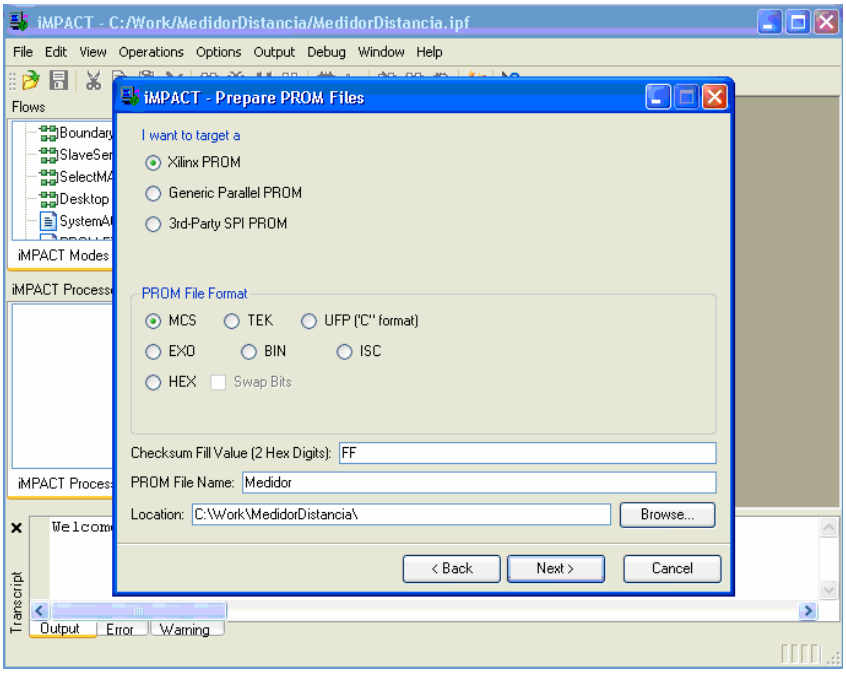


Figura 3.14: Ventana "Prepare PROM Files" de iMPACT

En la figura 3.14 se observa cómo se debe marcar la opción *Xilinx PROM*, la extensión *.mcs* para el formato del archivo a generar y cómo se va a nombrar a este archivo. Al pulsar sobre *Next*> de nuevo aparecerá el penúltimo

cuadro de diálogo antes de finalizar el proceso, según se muestra a continuación:

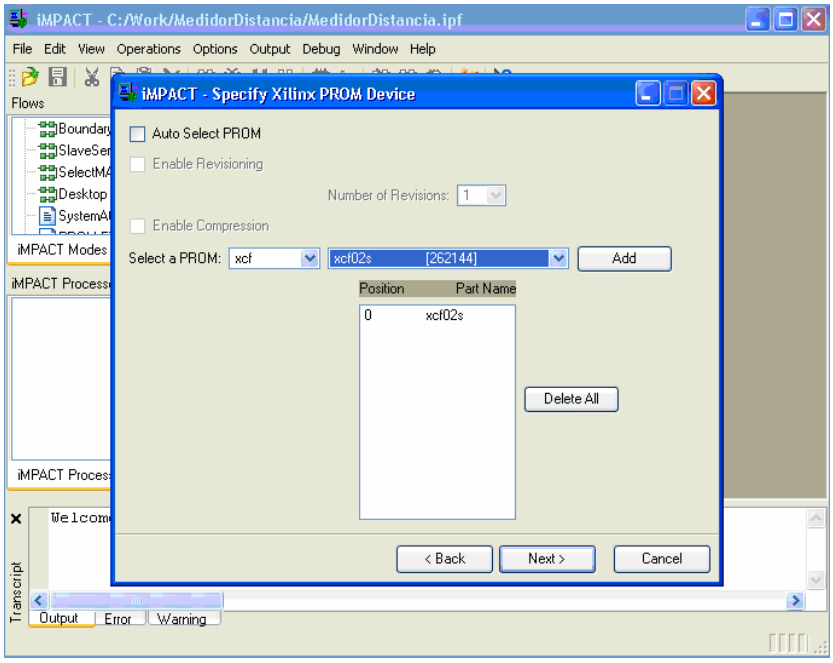


Figura 3.15: Ventana "Specify Xilinx PROM Device" de iMPACT

Esta figura 3.15 muestra las acciones a realizar sobre este cuadro. Estas acciones son la selección de la PROM *xcf02s*, que es la que lleva la placa de desarrollo Spartan III, y la validación de dicho modelo pulsando sobre el botón *Add*. Para terminar se pulsará de nuevo sobre el botón *Next>* y tras revisar el resumen que se ofrecerá posteriormente, se pulsará sobre *Finish*.

Al finalizar esta selección se mostrará un cuadro de diálogo para seleccionar el archivo *.bit* de programación generado anteriormente a introducir en el nuevo archivo *.mcs* para memoria Flash. Al introducir el archivo adecuado se comienza la generación y al finalizar se pregunta si se desea añadir nuevos archivos *.bit*; seleccionando no y pulsando sobre la opción de la izquierda llamada *Generate File* comienza la generación del archivo *.mcs*. Al finalizar la generación se ofrece la presentación que se observa en la figura 3.16:

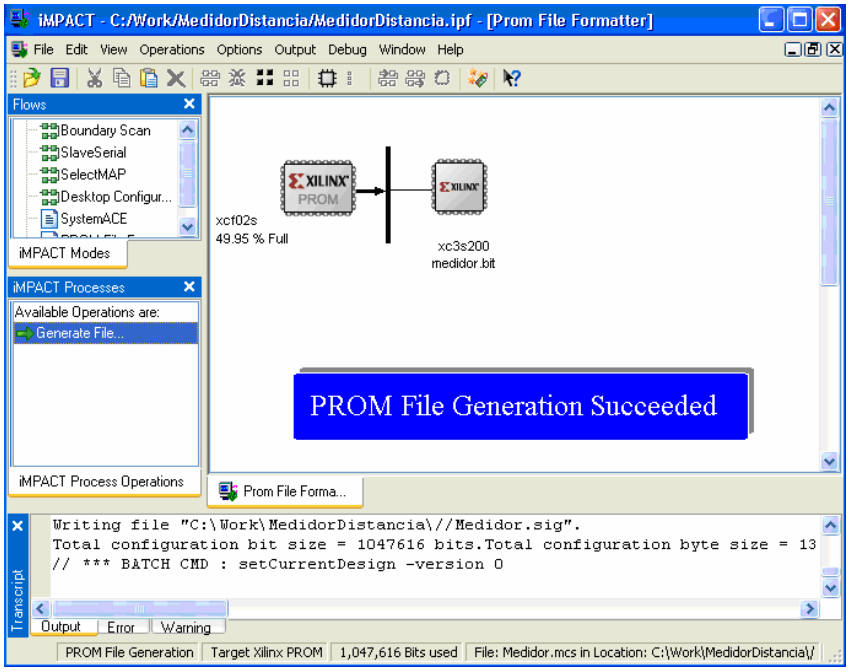


Figura 3.16: Finalización de generación de archivo PROM de iMPACT

Una vez se ha obtenido el archivo *.mcs* y habiendo obtenido el archivo *.bit*, estos pueden ser volcados sobre la placa uno a uno o ambos a la vez. Este proceso se realiza también con la aplicación iMPACT. Pero antes de nada es necesario generar un archivo con extensión *.msk* y con la misma información que el *.bit* manualmente.

Pulsando sobre el botón de la barra de herramientas del iMPACT, llamado *Launch Wizard*, se mostrará de nuevo el cuadro de diálogo de la figura 3.13. Sin embargo, en este caso se utilizará la primera opción.

Al pulsar sobre el botón *Finish* se detectarán automáticamente el tipo de programación y los dispositivos de la placa de desarrollo. Una vez se hayan detectado la FPGA y la memoria Flash se pedirán los archivos a introducir en uno y otro dispositivo.

Si solo se quiere programar la FPGA de forma volátil se seleccionará el archivo .bit y cuando se pida el archivo de la memoria Flash se pulsará sobre *Bypass*. Si no es ese el caso, se seleccionará el archivo *.msk* para la memoria Flash y configuración no volátil.

Cuando se termine la asignación de archivos a programar se pulsará con el botón derecho sobre el dispositivo a programar y se seleccionará la opción *Program* del menú desplegable. En ese momento se inicia la programación y si se obtiene una figura como la 3.17 significará que dicha programación ha sido correcta:

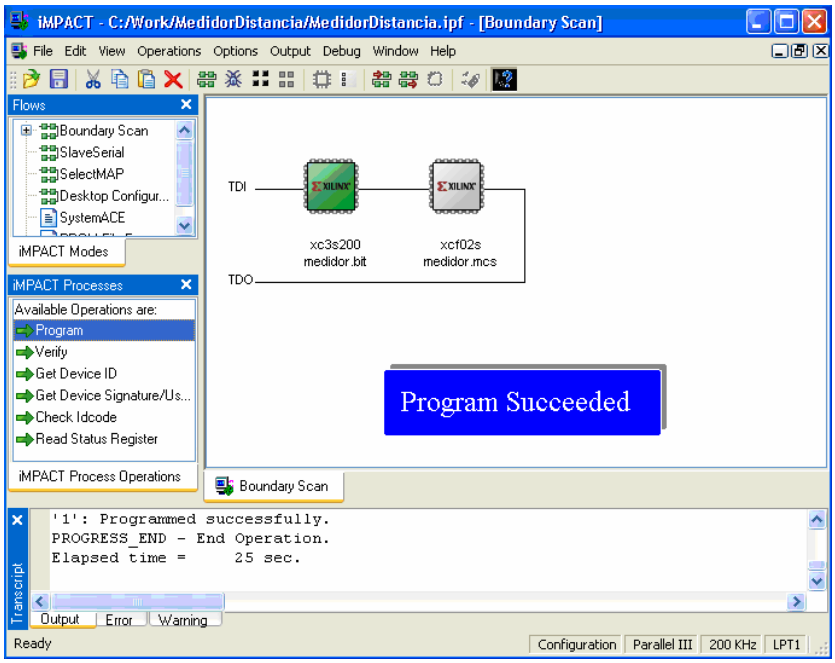


Figura 3.17: Finalización de generación de archivo FPGA de iMPACT

Para que la FPGA funcione con configuración volátil o con la memoria Flash habrá que modificar la posición ciertos jumpers de la placa de desarrollo Spartan III tal y como se especificó en el Capítulo Segundo.

Con esto concluye la explicación del funcionamiento del entorno de desarrollo ISE *WebPACK*™, desde el comienzo de la generación de código hasta la programación de la FPGA en la placa de desarrollo.

3.4. BLOQUES FUNCIONALES

En este momento se conocen ya todas las funciones que debe ser capaz de desempeñar el circuito codificado en VHDL. Por una parte se conocen las funciones de control y de comunicación con el usuario y por otra parte las de cálculo.

Según lo expuesto, el VHDL permite una filosofía Top-Down mediante la cual se pueden describir bloques y cajas negras incidiendo en su interior posteriormente. Este es el diseño que se ha seguido en la elaboración de este proyecto y el que se expone en esta memoria.

Es en este epígrafe donde se descenderán ordenadamente todos los niveles necesarios en la jerarquía del código VHDL. De esta manera se entenderá el funcionamiento total del sistema con todos sus detalles.

De tal modo, conociendo las especificaciones del diseño y de la placa de desarrollo Spartan III, se comentaron las funcionalidades para el circuito a codificar, es decir, los requisitos que ha de cumplir la FPGA. Según estos requisitos se diseñó una caja negra llamada Medidor (*Medidor.vhd*) sin prestar atención a su funcionamiento interno; esta es la entidad del circuito sintetizado en la FPGA.

A continuación se muestra la arquitectura correspondiente a esta entidad del circuito Medidor. Insistiendo en la jerarquía impuesta en este diseño, en la arquitectura de este circuito se podrán observar los diversos bloques funcionales con los que cuenta; que serán componentes de esta arquitectura así como entidades propias.

Dicha arquitectura se muestra esquematizada en la figura 3.18 que se expone, según lo comentado, a continuación:

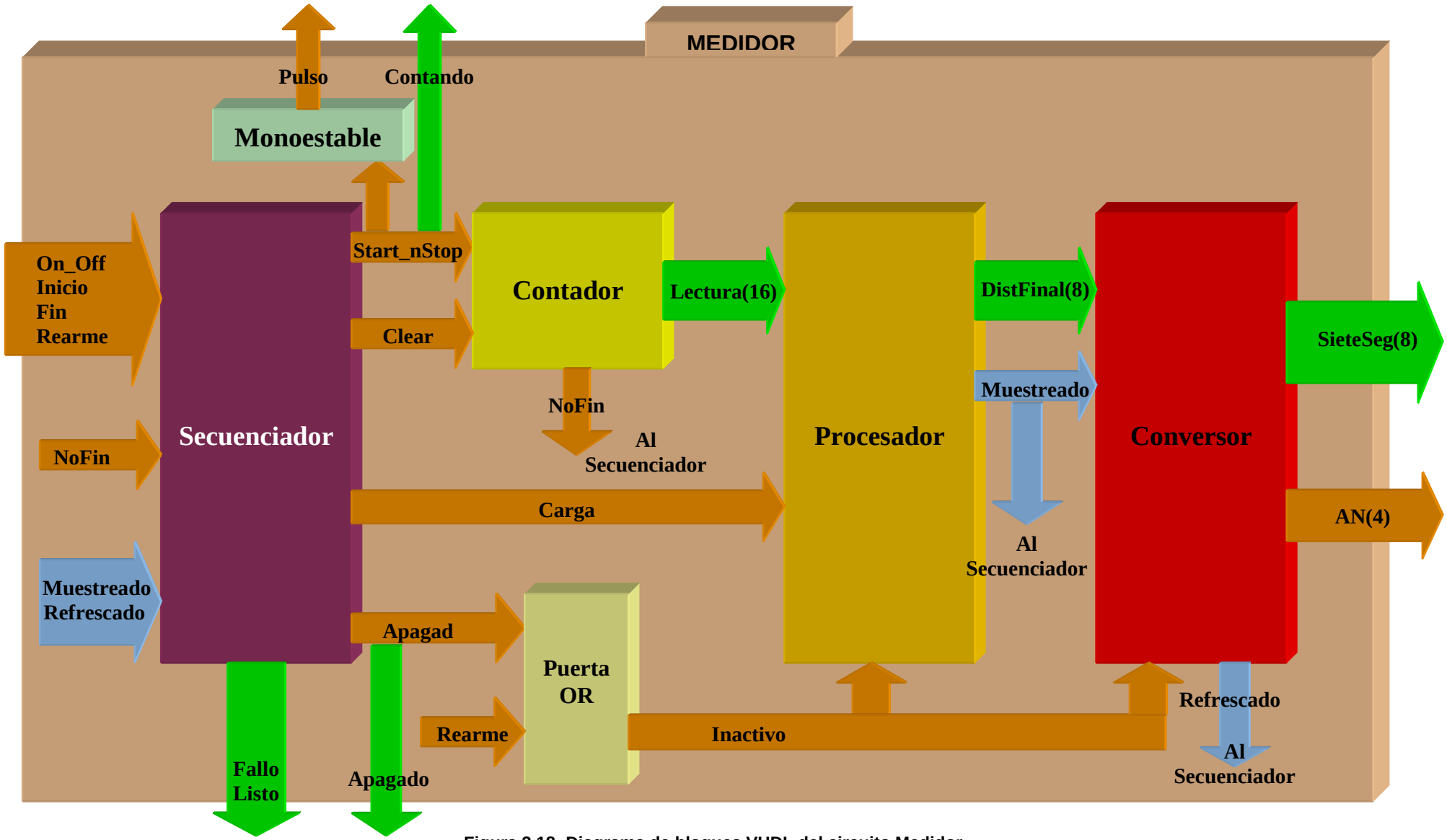


Figura 3.18: Diagrama de bloques VHDL del circuito Medidor

En cuanto a esta figura, se reseñará en primer lugar que, siendo la arquitectura de la entidad Medidor, las señales que entran y salen de ella coinciden con las de la figura 3.1; excepto la señal de reloj Clk que no está presente.

También es conveniente comentar que todos los bloques presentados en esta figura (excepto la Puerta OR) son componentes de la arquitectura del Medidor a la vez que entidades de los circuitos de menor jerarquía.

En la figura se detallan también todas las señales internas. La señal de reloj Clk se ha excluido para simplificar el esquema ya que llega a todos los bloques.

Para las señales comentadas se ha usado el mismo código de colores que en la figura 3.1: flechas azules son señales de sincronismo, flechas marrones son señales de control y flechas verdes son señales de datos e indicadores de estado. Esta asignación de colores se sigue en todas las figuras de los circuitos comentados en este capítulo.

A continuación se comentarán los componentes mostrados en la figura y la función que debe realizar cada uno:

- El **Secuenciador** es el bloque que se encarga de todo el control del sistema, gestiona todos los procesos y decide en cada momento qué se debe hacer. Por lo tanto, a este bloque le llegan todas las señales de control del Medidor así como señales internas de sincronismo y de control. El Secuenciador debe conocer el estado del resto de componentes para decidir cómo actuar, generando todas las señales de control hacia el resto de componentes y las de estado al exterior.

- El **Contador** es el siguiente bloque que, como su nombre indica, se ocupa de contar el tiempo de medida. Posee dos señales de control, que provienen del Secuenciador, llamadas *Start_nStop* para contar o parar y *Clear* para borrar la cuenta. De él salen la señal de control *NoFin*, para cuando la cuenta se desborda y no se ha parado el contador; y el vector de datos *Lectura(16)*, con el valor de la cuenta en cada momento. Cabe comentar que el número que aparece entre paréntesis en el nombre de cada vector es el número de bits que posee el mismo.

- El **Procesador** es el componente donde se tratan los datos del Contador, es decir *Lectura(16)*, para obtener el resultado adecuado. Este bloque además es el encargado de congelar el resultado con la señal Carga para permitir realizar una medida mientras se muestra el resultado anterior; activando la señal de sincronismo *Muestreado* al terminar de congelarlo. El vector de salida *DistFinal(8)* es el que contiene el dato de salida ya en términos de distancia en metros o centímetros.

- El componente **Conversor** es el encargado de toda la gestión del display de 4 dígitos, convirtiendo el dato de velocidad *DistFinal(8)* en 4 códigos de 7 segmentos que va multiplexando temporalmente con la señal *AN(4)*. Con la señal *Muestreado* que le llega del Procesador, calcula el refresco mínimo al que está ajustado, es decir, las veces que debe multiplexar en los 4 dígitos un mismo dato como mínimo. Cuando se alcanza el mínimo se activa la señal de sincronismo *Refrescado*.

- El componente **Monoestable** cumple la función que le da nombre, es decir que mientras que la señal *Star_nStop* (*Contando*) está activa durante todo el tiempo de la medida; este bloque activará la señal *Pulso* únicamente durante los 4 primeros milisegundos de cada medida.

- Por último, también se observa una **puerta OR***,* cuya misión es provocar el mismo efecto sobre los componentes Procesador y Conversor, cuando se pulse el *Rearme* y cuando se produzca la señal *Inactivo*.

El Medidor es el código de mayor nivel jerárquico y albergará al resto de circuitos. A continuación se muestra y comenta el código VHDL que da lugar al circuito Medidor tal y como se ha esquematizado en las figuras 3.1 y 3.18, es decir el archivo Medidor.vhd.

Antes de empezar con la descripción del código VHDL cabe destacar que, ya que el mismo se irá comentando fragmentadamente; la numeración de líneas que se adjunta con el código puede servir para observar la consecución del mismo. No obstante, el código total con todos los archivos .vhd de este proyecto serán incluidos en el DVD adjunto.

Se comienza por la descripción de las librerías y paquetes a utilizar junto con la entidad del Medidor. Este código VHDL se muestra a continuación:

```
1  LIBRARY IEEE;                    --Librerias
2     USE ieee.std_logic_1164.all;
3     USE ieee.std_logic_arith.all;
4
5
6  ENTITY Medidor IS                --Entidad Medidor
7     PORT (On_nOff, Inicio, Fin, Rearme, Clk      : in std_logic;   --Entradas y Salidas Medidor
8           Apagado, Contando, Pulso, Listo, Fallo : out std_logic;
9           SieteSeg                               : out std_logic_vector (7 DOWNTO 0);
10          AN                                     : out std_logic_vector (3 DOWNTO 0));
```

Lo primero que aparece es la declaración de librerías y paquetes que se usan específicamente en este código. La librería utilizada es la IEEE con los paquetes std_logic_1164 (incluye tipo de objeto std_logic) y std_logic_arith (incluye operaciones, conversiones y tipo unsigned).

Tras la declaración de librerías se describe la entidad del circuito Medidor, es decir, su encapsulado. Se observa como no existe ningún dato genérico, ya que no es necesario a este nivel. Por lo tanto se entra en la descripción de los puertos, es decir, entradas y salidas que se han presentado anteriormente.

En la arquitectura es donde se desarrolla todo el interior del circuito Medidor que se plasma en la figura. Este diseño se realiza, como se explicó anteriormente, comenzando por la parte declarativa de todos los elementos a usar posteriormente en el diseño. En este caso, habrá que declarar todos los componentes presentes en el Medidor. Esto se aprecia seguidamente:

```
14 ARCHITECTURE Bloques OF Medidor IS  --Bloques
15
16    COMPONENT Secuenciador              --Componente Secuenciador
17       PORT (On_nOff, Inicio, Fin, NoFin                    : in std_logic;
18             Muestreado, Refrescado, Clk, Rearme            : in std_logic;
19             Start_nStop, Clear, Carga, Apagad, Fallo, Listo : out std_logic);
20    END COMPONENT;
21
22    COMPONENT Contador                  --Componente Contador
23       GENERIC (NUMBITS  : POSITIVE := 16;
24                MAXIMO   : POSITIVE := 35200;
25                DIV_FREQ : POSITIVE := 50);
26       PORT (Start_nStop, Clear, Clk : in std_logic;
27             NoFin                   : out std_logic;
28             Lectura                 : out std_logic_vector ((NUMBITS-1) DOWNTO 0));
29    END COMPONENT;
30
31    COMPONENT Procesador                --Componente Procesador
32       GENERIC (NUMBITS_LECTURA, NUMBITS_MUESTRA : POSITIVE := 16;
33                NUMBITS_MICRO     : POSITIVE := 24;
34                NUMBITS_FINAL     : POSITIVE := 8);
35       PORT (Carga, Clk, Inactivo : in std_logic;
36             Lectura              : in std_logic_vector ((NUMBITS_LECTURA-1) DOWNTO 0);
37             Muestreado           : out std_logic;
38             DistFinal            : out std_logic_vector ((NUMBITS_FINAL-1) DOWNTO 0));
39    END COMPONENT;
40
41    COMPONENT Conversor                 --Componente Conversor
42       GENERIC (NBITS : POSITIVE := 8);
43       PORT (DistFinal       : in std_logic_vector ((NBITS-1) DOWNTO 0);
44             Clk, Inactivo : in std_logic;
45             Muestreado    : in std_logic;
46             Refrescado    : out std_logic;
47             AN            : out std_logic_vector (3 DOWNTO 0);
48             SieteSeg      : out std_logic_vector (7 DOWNTO 0));
49    END COMPONENT;
```

En el fragmento de código se observa la declaración de todos los componentes que se han descrito. Cada componente declarado tiene unos datos genéricos y una descripción de puertos; describiendo la entidad del nivel inferior. Estos componentes son declarados de forma idéntica a la que la entidad correspondiente es descrita.

Los datos genéricos que aparecen en ellos sirven para poder ajustar, variar el diseño de forma fácil y fiable, y para evitar errores. Además con estos datos genéricos los módulos tendrán mayor grado de adaptabilidad para una futura reutilización. Estos datos genéricos serán comentados en cada bloque con más detalle.

En la descripción de puertos, se observan todas las señales que entran y salen a cada uno de los componentes en la figura 3.18, con idénticos nombres para facilitar la relación del esquema y del código VHDL. Para verificar la longitud de los vectores, en ocasiones hay que observar los datos genéricos que la asignan. Es importante hacer mención a que, el orden en el que posteriormente se asignarán las señales a los componentes en la parte descriptiva de la arquitectura; es igual al descrito en los puertos de los componentes en la parte declarativa.

A continuación se declaran todas las señales necesarias para relacionar los componentes que son las siguientes:

```
56
57    SIGNAL Start_nStop, Clear, Carga, NoFin : std_logic;  --Conexion entre Componentes
58    SIGNAL Apagad, Inactivo, Muestreado, Refrescado : std_logic;
59    SIGNAL Lectura : std_logic_vector (15 DOWNTO 0);
60    SIGNAL DistFinal : std_logic_vector (7 DOWNTO 0);
61
```

A este nivel de jerarquía, todas las señales declaradas se utilizan para conectar los componentes o para generar señales que se usen internamente y a la vez sean de salida. Es por esto que las señales declaradas se pueden localizar en la figura 3.18.

Lo siguiente que se debe generar en el código VHDL del medidor es la parte descriptiva de la arquitectura, esto es siempre lo que se encuentra entre el primer BEGIN y el último END. Dicha descripción se realiza como se indica en el siguiente código:

```
61
62      BEGIN
63         S_Control : Secuenciador PORT MAP                    --Instancia Secuenciador
64            (On_nOff, Inicio, Fin, NoFin, Muestreado, Refrescado, Clk,
65             Rearme, Start_nStop, Clear, Carga, Apagad, Fallo, Listo);
66
67         Temporizador : Contador PORT MAP                     --Instancia Contador
68            (Start_nStop, Clear, Clk, NoFin, Lectura);
69
70         Procesad : Procesador PORT MAP                       --Instancia Procesador
71            (Carga, Clk, Inactivo, Lectura, Muestreado, DistFinal);
72
73         Convert : Conversor PORT MAP                         --Instancia Conversor
74            (DistFinal, Clk, Inactivo, Muestreado, Refrescado, AN, SieteSeg);
75
76         Monoest : Monoestable PORT MAP                       --Instancia Monoestable
77            (Start_nStop, Clk, Pulso);
78
79         Inactivo <= Rearme OR Apagad;                        --Multifuncion de Señal Inactivo
80
81         Apagado <= Apagad;                                   --Asignacion Señales a Exterior
82         Contando <= Start_nStop;
83
84   END Bloques;
```

De tal forma, cada componente se instancia aplicándole una etiqueta y nombrándole. Posteriormente se le hace la asignación de señales pertinente, siempre en el mismo orden que en la declaración.

En el código mostrado también se aplica una puerta OR para conseguir la señal inactivo (como en el esquema) y se asignan señales para poder ser utilizadas en el exterior. Esta asignación no es trivial; debe realizarse porque no es posible que una señal que entre a un componente salga al exterior, aunque si es posible que una señal entre a dos componentes distintos.

Como se aprecia en el código VHDL mostrado, cada operación realizada se identifica con un comentario siempre precedido del símbolo --. Estos comentarios sirven para aclarar y seguir el código. En esta memoria también se utilizarán para hacer referencia a las partes del código que se estén comentando, escribiendo el comentario entre paréntesis de la siguiente manera: (--Comentario).

Con todo el código VHDL expuesto y comentado, se da por finalizada la descripción del circuito VHDL sintetizado de mayor nivel jerárquico, el medidor. Como se ha comprobado, la misión de este circuito no es otra que la de unir y conectar el resto de circuitos de menor nivel (componentes) para obtener un código único, simple y compacto. A continuación se comentarán en detalle los componentes que conforman el medidor.

3.4.1. Secuenciador

Este componente, como ya se ha introducido, es el que posee toda la “inteligencia” del sistema, siendo un elemento puramente secuencial; de ahí su nombre.

Dependiendo del valor de sus entradas generarán unas salidas adecuadas a la situación en cada momento. Las entradas podrán ser externas o elementos de sincronismo o control diseñadas al efecto. Las salidas serán elementos de control o indicadores para el usuario.

Por lo tanto, este componente tiene como misión el control del resto de componentes; que son los que verdaderamente actúan y que informan de su estado al controlador. Este componente constituye un circuito secuencial denominado autómata.

El entorno de desarrollo utilizado posee diversos métodos para el diseño de autómatas, e incluso diversas filosofías de síntesis. Sin embargo, para no variar el lenguaje de programación en el proyecto, el diseño del autómata que compone el *Secuenciador* está codificado en VHDL.

Dado que el lenguaje VHDL tiene ciertas estructuras rígidas para llegar a sintetizar determinados circuitos, como los autómatas, se utilizará una de estas estructuras para el diseño del Secuenciador. Todo este código se comentará a continuación y se encuentra en el archivo *Secuenciador.vhd*.

La declaración de librerías y la descripción del encapsulado se compone de este código:

```
1  LIBRARY IEEE;                    --Librerias
2     USE ieee.std_logic_1164.all;
3
4
5  ENTITY Secuenciador IS           --Entidad Secuenciador
6     PORT (On_nOff, Inicio, Fin, NoFin                    : in std_logic;   --Entradas y Salidas
7           Muestreado, Refrescado, Clk, Rearme            : in std_logic;
8           Start_nStop, Clear, Carga, Apagad, Fallo, Listo : out std_logic);
9  END Secuenciador;
10
```

En cuanto a las librerías, cabe mencionar que para este diseño solo se utilizará el paquete std_logic_1164.

Aunque en la figura 3.18 y en su comentario se han expuesto todas las señales del sistema, en cada componente se enumerarán las señales que entran y salen de él así como su función en él y en el sistema.

De tal forma y exceptuando las señales Clk y Rearme, a continuación se comentarán las señales de entrada al bloque Secuenciador:

- *On_nOff:* Esta señal pondrá en standby el sistema cuando esté a nivel bajo; dejando el display inactivo, borrando toda la información de medidas anteriores y parando cualquier cuenta de tiempo.

- *Inicio:* Cuando esta señal se encuentra a nivel alto permitirá el inicio de una nueva medición siempre y cuando se haya refrescado un número mínimo de veces el display.

- *Fin:* El significado de esta señal es de índole física y se genera en la placa auxiliar. Cuando el eco de la señal de ultrasonidos producida en dicha placa regrese a la misma esta señal se encontrará a nivel alto.

- *NoFin:* Esta señal proviene del interior del Medidor, concretamente del bloque Contador. Cuando se activa a nivel alto significa que el Contador se ha desbordado y todavía no se le ha ordenado parar.

- *Muestreado:* Siempre y cuando un dato sea correctamente guardado en un registro que posee el bloque Procesador, esta señal se encontrará a nivel alto hasta que se pretenda guardar otro dato.

- *Refrescado:* Cuando una medida nueva sea refrescada en el Conversor tantas veces en sus cuatro dígitos como un dato genérico indique, esta señal se activará a nivel alto hasta que la cifra vuelva a cambiar.

Las señales de salida de este circuito Secuenciador son las que actuarán como control en el resto del Medidor y se comentan a continuación:

- *Start_nStop:* Esta señal activa la cuenta del componente Contador cuando se encuentra a nivel alto y la desactiva a nivel bajo.

- *Clear:* Cuando esta señal se pone a nivel alto, el valor de la cuenta que tiene almacenado el contador es borrado.

- *Carga:* Esta es la señal que activa la carga del registro del Procesador cuando se encuentra a nivel alto.

- *Apagad:* Cuando el sistema se encuentra en standby esta señal se pone a nivel alto para detener y resetear el resto de componentes del sistema.

- *Fallo:* Si el sistema se encuentra fuera de control, se activa esta señal.

- *Listo:* Mientras una medición es refrescada el número mínimo de veces y se activa Inicio, esta señal se pone a nivel alto antes de una medida.

A continuación se describirá la arquitectura del Secuenciador. Se sabe que la estructura que hay que implementar es un autómata así como la función de entradas y salidas y la funcionalidad global del bloque. Por lo tanto lo siguiente es reflexionar sobre el diseño del autómata antes de comenzar a codificar en VHDL.

El autómata que se va a implementar es de tipo Moore, dado que para la aplicación especificada es más sencillo de utilizar y no supondrá ninguna ventaja usar un autómata Mealy. Dichos autómatas se componen de dos clases de elementos: estados y transiciones.

Según lo dispuesto, el diseño a implementar seguirá un diagrama que rige el funcionamiento del autómata, compuesto de estados (Cuadrados) y transiciones (Flechas). Así se genera la siguiente figura 3.19:

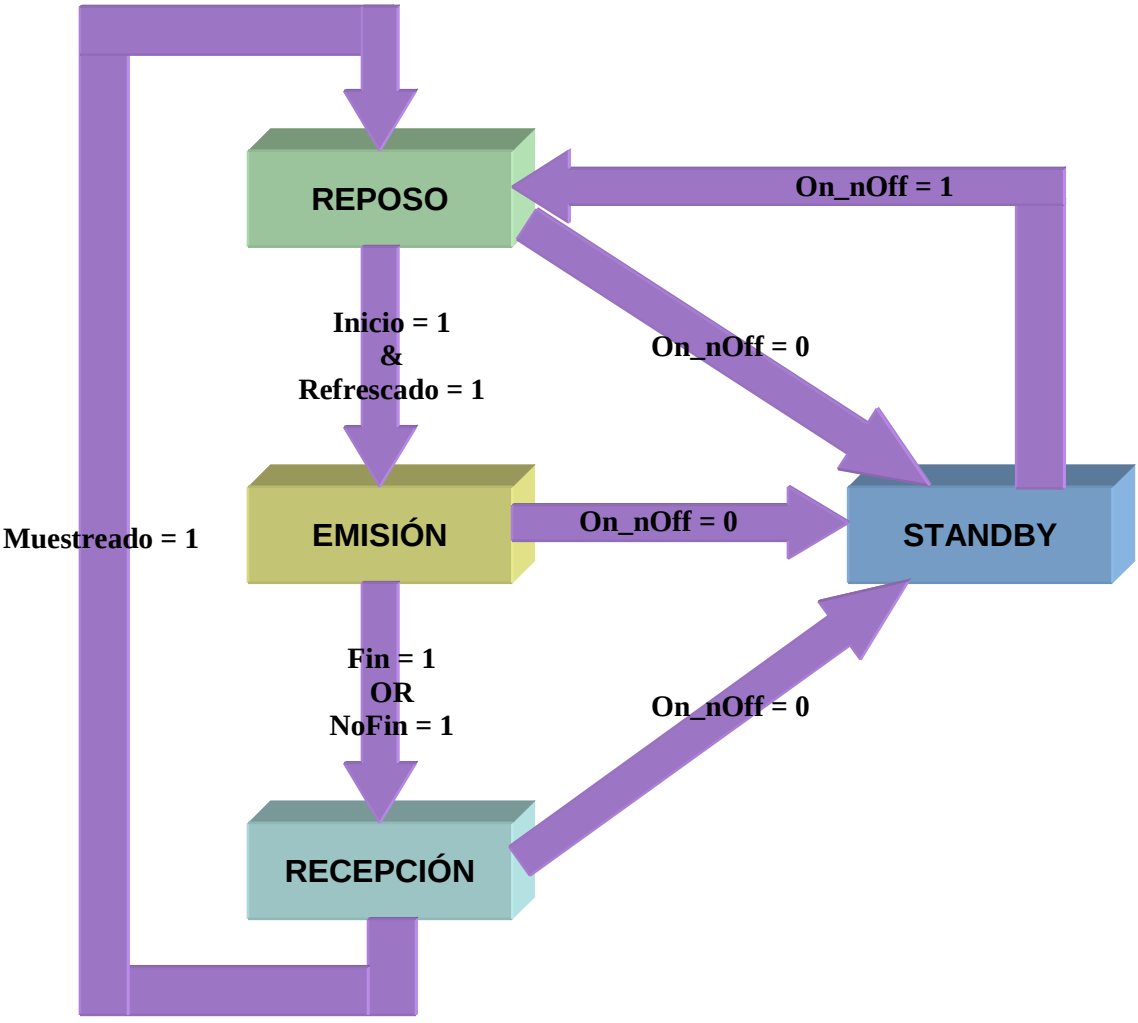


Figura 3.19: Diagrama de estados del circuito Secuenciador

Siguiendo el diagrama de esta figura se consigue que el autómata funcione correctamente y cumpla con las funcionalidades necesarias para el Secuenciador. Además en la figura se observa que son necesarios cuatro estados, es decir, cuatro asignaciones de salidas distintas posibles, que son las siguientes: *STANDBY, REPOSO, EMISIÓN y RECEPCIÓN*.

Cada uno de estos estados asignará ciertos valores a las salidas del circuito para obtener la funcionalidad deseada. Esto se observa en la siguiente tabla:

Asignación de Salidas

Señales / Estado	***Start_nStop***	***Clear***	***Carga***	***Apagad***	***Fallo***	***Listo***
REPOSO	0	1	0	0	0	1
EMISIÓN	1	0	0	0	0	0
RECEPCIÓN	0	0	1	0	0	0
STANDBY	0	1	0	1	0	0
OTHERS	0	1	0	1	1	0

Tabla 3.2: Asignación de salidas según estados

Para conectar y relacionar todos los conceptos expuestos acerca del autómata implementado se resumirá la secuencia del proceso de medida simulando una medición real.

Partiendo del estado STANDBY, el autómata pasará al estado REPOSO cuando la señal On_nOff se ponga a nivel alto. En el estado de REPOSO se encenderá el display poniendo a nivel bajo la señal Apagad, y se pondrá a nivel alto la señal Clear y a nivel alto la señal indicadora Listo.

Cuando la señal Inicio se active a nivel alto se pasará al estado EMISIÓN. En este estado se emitirá una onda de ultrasonidos y se comenzará la cuenta del tiempo de eco mediante la activación a nivel alto de la señal Start_nStop.

Si la onda de ultrasonidos es recibida o si no se pierde y nunca vuelve o no es detectada se activarán las señales Fin o NoFin, respectivamente. En este caso se pasará del estado de EMISIÓN al de RECEPCIÓN. En este estado el Procesador tratará el resultado del Contador y lo guardará en un registro mediante la puesta a nivel alto de la señal Carga.

En el momento en el que el registro del Procesador haya guardado el dato, la señal Muestreado se pondrá a nivel alto y el Conversor comenzará su función mostrando en el display el resultado de la medida. De tal forma, en este momento se pasará al estado de REPOSO de nuevo preparando el sistema para una nueva medición.

De tal modo, si la señal Inicio sigue activa, en cuanto se refresque el display un número de veces estipulado, la señal Refrescado se actualizará a nivel alto y se comenzará inmediatamente una nueva medición. Si la señal Inicio se encuentra a nivel bajo, cuando la señal Refresco se encuentre a nivel alto se permanecerá indeterminadamente en el estado REPOSO.

Por otra parte, desde cualquier estado, si se pone a nivel bajo la señal On_nOff, se pasará al estado de STANDBY desactivando el sistema. En este estado se resetearan todos los registros y se apagará el display con las señales Clear y Apagad a nivel alto.

Así, queda comprobado que es posible diseñar el autómata en función del diagrama y la tabla expuestos. A continuación se comenzará a introducir el código VHDL necesario para implementar un autómata de las características y estructura expuestas. Seguidamente se muestra el inicio de la arquitectura Autómata dentro del Secuenciador y su parte declarativa:

```
11
12  ARCHITECTURE Automata OF Secuenciador IS  --Automata
13
14        TYPE ESTADO IS (REPOSO, EMISION, RECEPCION, STANDBY); --Definicion 4 Estado
15        SIGNAL Actual, Siguiente : ESTADO;                    --Declaracion Señales Estados
```

Lo primero que se debe comentar es la creación de un tipo de señal generado llamado ESTADO, que tendrá cuatro valores posibles coincidiendo con los estados del autómata presentados en la figura 3.19. Además se declaran dos señales de este tipo, son Actual y Siguiente. Esto se hace para poder establecer el sistema secuencial en el que se basa el autómata sabiendo de qué estado viene para saber donde irá, es decir, para realizar las transiciones y asignar los estados.

En un autómata codificado en VHDL siempre existirán dos partes diferenciadas: una puramente secuencial (--Proceso Secuencial) que seguirá el cambio de reloj y otra combinacional (--Proceso Combinacional) que reaccionará a cambios en las entradas. A continuación se establece el inicio de la parte descriptiva de la arquitectura comenzando con la parte secuencial del autómata en una estructura ya comentada; el llamado proceso. Obsérvese el siguiente código VHDL:

```
16
17    BEGIN
18       PROCESS (Clk, Rearme)                  --Proceso Secuencial
19
20          BEGIN
21             IF Rearme = '1' THEN                --Rearme Asincrono
22                Actual <= REPOSO;
23             ELSIF Clk 'EVENT AND Clk = '1' THEN --Asignacion de Cambio de Estado
24                Actual <= Siguiente;
25             END IF;
26
27          END PROCESS;                          --Fin de Proceso Secuencial
```

Aquí se puede apreciar como este proceso o parte secuencial del autómata solamente reacciona a cambios en el reloj Clk, o a la señal de reseteo asíncrono llamada Rearme.

Si la señal Rearme se activa a nivel alto el autómata se irá al estado REPOSO mediante la asignación de dicho valor a la señal Actual. En caso contrario, si se diese un flanco positivo de reloj el valor guardado en la señal Siguiente pasaría a la señal Actual; produciéndose un cambio de estado o no en función de la parte combinacional del autómata que se comenta seguidamente:

```
28
29       PROCESS (Actual, On_nOff, Inicio, Fin, NoFin, Muestreado, Refrescado, Rearme)
30
31          BEGIN                                 --Proceso combinacional
32             CASE Actual IS
```

En este fragmento de código se observa la declaración de un nuevo proceso, el correspondiente a la parte combinacional del autómata. Como ya se adelantó, este proceso tiene en su lista de sensibilidad todas las entradas del Secuenciador junto con la señal Actual.

También se observa la definición de una sentencia de tipo *CASE* que actúa sobre la señal Actual. Esta orden es semejante a un selector que, dependiendo del valor asignado a la señal Actual, ejecutará un código u otro.

De tal forma se consigue que, dependiendo del estado que haya en cada momento, es decir, del valor de la señal Actual; se establezca cual es la transición posible mediante la asignación de la señal Siguiente así como todas las salidas.

En adelante se comentará uno a uno cada código para cada estado que se encuentra dentro de la sentencia CASE. Tanto las transiciones como la asignación de salidas fueron comentadas al exponer la figura 3.19 y la tabla 3.2 y se podrá verificar su implementación.

Sin más se comienza a describir el estado REPOSO:

```
33
34              WHEN REPOSO =>                  --Estado Inicial Reposo
35                 IF On_nOff = '0' THEN           --Def. Sig. Estado
36                    Siguiente <= STANDBY;
37                 ELSIF Inicio = '1' AND Refrescado = '1'THEN
38                    Siguiente <= EMISION;
39                 ELSE
40                    Siguiente <= REPOSO;
41                 END IF;
42                 Clear <= '1';                   --Asignacion Señales
43                 Start_nStop <= '0';
44                 Carga <= '0';
45                 Apagad <= '0';
46                 Listo <= '1';
47                 Fallo <= '0';
```

En este fragmento de código puede observarse que, para cada estado, efectivamente se pueden diferenciar dos partes. La primera es donde se definirá la transición con sus condiciones y se asigna el valor del estado Siguiente según lo expuesto en la figura 3.19 (--Def. Sig. Estado). Posteriormente se asignan todas las salidas con los valores especificados según la tabla 3.2 (--Asignación Señales).

Este comentario se centrará en la asignación de las salidas, que siempre se dará en el mismo orden.

En este estado, en el cual comienza otro ciclo, se debe dejar todo listo para volver a realizar una medición. Por esto, de todas las salidas, se activan el Clear del Contador para su reseteo y el indicador Listo hacia el usuario. El resto de salidas permanecerán a nivel bajo.

El siguiente estado a describir es EMISIÓN:

```
48
49              WHEN EMISION =>                 --Estado de Cuenta-Emision
50                 IF On_nOff = '0' THEN           --Def Sig Estado
51                    Siguiente <= STANDBY;
52                 ELSIF Fin = '1' OR NoFin = '1' THEN
53                    Siguiente <= RECEPCION;
54                 ELSE
55                    Siguiente <= EMISION;
56                 END IF;
57                 Clear <= '0';                   --Asignacion Señales
58                 Start_nStop <= '1';
59                 Carga <= '0';
60                 Apagad <= '0';
61                 Listo <= '0';
62                 Fallo <= '0';
```

En este estado, se debe emitir una onda de ultrasonidos y se debe comenzar la cuenta de tiempo de eco de la misma. Esto se realiza mediante la puesta a nivel alto solamente de la señal Start_nStop que llegará al Contador, al Monoestable y saldrá al exterior como el indicador Contando.

El siguiente código pertenece al estado de RECEPCIÓN:

```
63
64                WHEN RECEPCION =>                --Estado de Registro Recep
65                   IF On_nOff = '0' THEN            --Def Sig Estado
66                      Siguiente <= STANDBY;
67                   ELSIF Muestreado = '1' THEN
68                      Siguiente <= REPOSO;
69                   ELSE
70                      Siguiente <= RECEPCION;
71                   END IF;
72                   Clear <= '0';                    --Asignacion Señales
73                   Start_nStop <= '0';
74                   Carga <= '1';
75                   Apagad <= '0';
76                   Listo <= '0';
77                   Fallo <= '0';
```

El estado de RECEPCIÓN consiste en la sucesión del evento de recepción del eco de la onda de ultrasonidos. Por lo tanto, en este momento se debe parar el Contador inmediatamente. De tal forma, la única señal que debe estar a nivel alto en este estado es la de Carga que almacene la medida de distancia en un registro del componente Procesador.

Para el estado de STANDBY se codifica lo siguiente:

```
78
79                WHEN STANDBY =>                  --Estado de Standby
80                   IF On_nOff = '1' THEN            --Def Sig Estado
81                      Siguiente <= REPOSO;
82                   ELSE
83                      Siguiente <= STANDBY;
84                   END IF;
85                   Clear <= '1';                    --Asignacion Señales
86                   Start_nStop <= '0';
87                   Carga <= '0';
88                   Apagad <= '1';
89                   Listo <= '0';
90                   Fallo <= '0';
```

Este estado de mantener todo el sistema inactivo y resetear todos los elementos de almacenamiento. Por esto se encuentran a nivel alto la señal Clear, que borra el Contador, y la señal Apagad, que resetea el registro del Procesador y neutraliza al Conversor apando el display.

El siguiente código no pertenece a ningún estado del diagrama de la figura 3.19, pero es una práctica saludable el establecer un caso alternativo si la ejecución sale de alguno de los anteriores estados. Aunque teóricamente esto es imposible, en VHDL suele aparecer este código.

```
91
92                WHEN OTHERS =>                   --Resto de Casos
93                   Clear <= '1';                    --Asignacion Señales
94                   Start_nStop <= '0';
95                   Carga <= '0';
96                   Apagad <= '1';
97                   Listo <= '0';
98                   Fallo <= '1';
99
100            END CASE;
101
102         END PROCESS;                        --Fin de Proceso Combinacional
103
104   END Automata;
```

Como se observa, en este código se resetean todos los registros y se inactiva el display como si se tratase del estado STANDBY. Sin embargo, este estado residual es el único donde se pone a nivel alto la señal indicadora Fallo. Por lo tanto, si se llegase a caer en este estado el usuario tendría cuenta de ello y resetearía el sistema con la señal Rearme.

Finalmente, en el código anterior también se observa como se finaliza la sentencia CASE, como se cierra el proceso combinacional donde se han desarrollado todos los estados y, por último, como se termina la arquitectura Autómata del Secuenciador.

Con este código queda descrito totalmente el Secuenciador, corazón del sistema Medidor implementado en la FPGA. De tal modo se ha conseguido la funcionalidad deseada para poder secuenciar un proceso de medida de distancia cíclico, su detención y su apagado.

3.4.2. Contador

El *Contador* será el componente dedicado a llevar la cuenta de tiempo que tarda una onda de ultrasonidos en recorrer el espacio que dista entre la fuente y el obstáculo más cercano; y volver al receptor de ultrasonidos.

Esta cuenta será realizada en microsegundos para obtener una precisión adecuada a lo largo de la cadena que conseguirá dar el resultado de la distancia. Además también se tendrán controles que permitirán al bloque Secuenciador actuar sobre el Contador y recibir información sobre su desborde.

El código que se ha generado para conseguir toda esta funcionalidad se encuentra en el archivo *Contador.vhd* que se comentará a continuación. Como siempre, se empezará por las librerías y entidad:

```
1  LIBRARY IEEE;                    --Librerias
2     USE ieee.std_logic_1164.all;
3     USE ieee.std_logic_arith.all;
4
5
6  ENTITY Contador IS               --Entidad Contador
7     GENERIC (NUMBITS  : POSITIVE := 16;          --Numero de Bits
8              MAXIMO   : POSITIVE := 35200;       --Desbordamiento de Contador
9              DIV_FREQ : POSITIVE := 50);         --Division de Frec. Reloj
10    PORT (Start_nStop, Clear, Clk : in std_logic;   --Entradas y Salidas
11          NoFin                   : out std_logic;
12          Lectura                 : out std_logic_vector ((NUMBITS-1) DOWNTO 0));
13 END Contador;
14
```

Como se observa, los paquetes declarados son el std_logic_1164 (tipo std_logic) y el std_logic_arith (tipo Unsigned y conversiones), de la librería IEEE.

En cuanto a la entidad, se pueden observar varios datos genéricos cuya función se comenta a continuación:

- *NUMBITS:* Este dato proporciona en todo el código el número de bits de la señal de salida llamada Lectura y todo lo relacionado con ella. Estos vectores tendrán una longitud de 16 bits.

- *MAXIMO:* Cuando el contador no recibe orden de parar la cuenta, llegará a un máximo especificado en este dato genérico, en este caso el máximo es 35200 microsegundos. Este dato corresponde a una distancia de 6054400 micrómetros. Ello supone más de 6 metros que es lo máximo a lo que llega un dispositivo estándar de ultrasonidos y lo fijado en especificaciones del medidor.

- *DIV_FREQ:* Este será un dato genérico muy común, ya que, los circuitos tienen que adaptar la frecuencia que les proporciona el oscilador de la placa (50 MHz.) a la frecuencia más adecuada. En este caso, la división de frecuencia tiene un valor de 50, lo que quiere decir que la frecuencia de funcionamiento final del bloque será de 1 MHz.

Seguidamente se encuentran las entradas y salidas del circuito. Las entradas de este bloque, según lo expuesto en la figura 3.17 y en esta entidad, son las siguientes:

- *Start_nStop:* Esta señal provocará cuando se tenga a nivel alto que la cuenta de este componente se incremente cada microsegundo en uno. Si esta señal se encuentra a nivel bajo, la cuenta será paralizada en el valor que tuviera.

- *Clear:* Dicha señal limpiará el valor de cuenta que se tenga únicamente cuando se encuentre a nivel alto. De tal forma, dejará el Contador a cero para una nueva medida.

Las salidas de este componente llamado Contador se exponen a continuación:

- *Lectura(16):* Este vector de salida es que el contiene el dato del tiempo de eco contado en microsegundos y actualizado en el

momento. Su longitud se ajusta mediante el dato genérico NUMBITS y para este proyecto será de 16 bits.

- *NoFin:* Esta es una señal de control que se genera para indicar al Secuenciador que se ha alcanzado el dato genérico MAXIMO (35200 microsegundos) sin que se reciba onda de ultrasonidos de eco. Por lo tanto se debe dejar de esperar la misma.

A continuación se codifica la arquitectura del Contador llamada *Nbits*. Esta arquitectura carece de parte declarativa. Por lo tanto, se muestra directamente la parte descriptiva de esta arquitectura:

```
16  ARCHITECTURE Nbits OF Contador IS          --Nbits
17        BEGIN
18        PROCESS (Clk, Start_nStop, Clear)      --Proceso de Cuenta
19
20           VARIABLE c : UNSIGNED ((NUMBITS-1) DOWNTO 0);   --Variable de Cuenta
21           VARIABLE d : INTEGER RANGE 0 TO DIV_FREQ;       --Variable Div. Frec.
22
23           BEGIN
24              IF Clk 'EVENT AND Clk = '0' THEN    --Flanco descendente de Clk
25
26                 IF Start_nStop = '1' THEN           --Señal de Cuenta
27
28                    IF d = DIV_FREQ THEN                --Desborde Cuenta Primaria(Div)
29                       d:= 0;                              --Reinicio C. Primaria
30
31                    IF c = MAXIMO THEN                  --Cuenta Secundaria(Real)
32                       NoFin <= '1';                       --Desbordamiento
33
34                    ELSE                                   --Cuenta Habitual
35                       NoFin <= '0';
36                       c := c + 1;
37
38                    END IF;
39
40                 ELSE                                --Incremento C. Primaria(Div)
41                    d := d + 1;
42
43                 END IF;
44
45              ELSIF Clear = '1' THEN              --Clear Sincrono
46                 d := 0;
47                 c := (OTHERS => '0');
48                 NoFin <= '0';
49
50              END IF;
51
52           END IF;
53
54        Lectura <= STD_LOGIC_VECTOR (c);       --Conversion a vector binario
55
56        END PROCESS;                        --Fin Proceso de Cuenta
57  END Nbits;
```

Lo primero que se aprecia en esta arquitectura es la descripción de un proceso para la cuenta (--Proceso de Cuenta). Su lista de sensibilidad será la señal de reloj Clk, la señal Start_nStop, y la señal Clear. Esto quiere decir que, solamente cuando alguna de las tres señales cambien, se ejecutará el proceso.

En la parte declarativa del proceso se definen dos variables para conseguir realizar el proceso de cuenta. La variable c lleva la cuenta habitual y es de tipo UNSIGNED de la misma longitud que la señal de salida. La variable d es la que lleva la cuenta primaria para la división de frecuencia de cuenta y es de tipo INTEGER con valores posibles de 0 al dato genérico DIV_FREQ.

Posteriormente comienza la parte descriptiva del proceso con la sentencia IF en la cual se entra únicamente si en la señal de reloj Clk se da un flanco negativo. Este flanco es opuesto al del Secuenciador. Al igual que sucederá con el resto de componentes, esto se hace para que el sistema sea más rápido y eficiente al eliminar los efectos del retardo de señales y acotarlos en menos de medio ciclo de reloj.

Dentro de esta secuencia de sincronismo se encuentra un bloque bajo el dominio de la condición de la señal Start_nStop = 1.

Si es ese el caso y además la variable de cuenta primaria d es igual al dato de división de frecuencia, habrá que reiniciar dicha variable e incrementar en uno la variable de cuenta habitual c (--Desborde Cuenta Primaria(Div)). Esto se realizará siempre y cuando la variable c no sea el número de cuenta máximo; en caso contrario se pondrá a nivel alto la señal NoFin y no se incrementará la variable c(--Cuenta Secundaria(Real)).

En caso contrario, si la variable de cuenta primaria no es igual al dato de división de frecuencia, esta variable se incrementará (--Incremento C. Primaria (Div)).

De esta forma se divide la frecuencia del oscilador en placa incrementando la cuenta de salida en 1 cuando se produzcan 50 ciclos de la frecuencia original. Esto quiere decir que se cuenta a 1 MHz., o lo que es lo mismo, cada microsegundo.

Si la señal Start_nStop se encuentra a nivel bajo y la señal Clear está a nivel alto, se ponen a cero las variables c y d y se pone a nivel bajo la señal NoFin (--Clear Síncrono).

Si no se diese el flanco negativo de la señal Clk no se realiza ninguna acción en este proceso.

Antes de finalizar el proceso de cuenta, se asigna a la señal de salida Lectura(16) la variable de cuenta habitual c(16) mediante una conversión a tipo std_logic.

Por último, acaba el proceso y la arquitectura del componente Contador llamada Nbits.

De tal modo, queda descrito el código VHDL que sintetiza el dispositivo de cuenta en el sistema Medidor.vhd de la FPGA, es decir, el Contador.

3.4.3. Procesador

Este es el circuito del sistema Medidor que permite el tratamiento de datos para obtener la distancia y ajustarla a valores que puedan ser mostrados en dos precisiones.

Además, este circuito es el que permite que se mantenga en el display la distancia medida anteriormente mientras que no se finalice una nueva medición ni se apague el sistema.

Este sistema se estructurará, jerárquicamente en tres componentes para dividir en tareas más sencillas la función que realiza este componente. Dicha estructura se muestra en la siguiente figura 3.20:

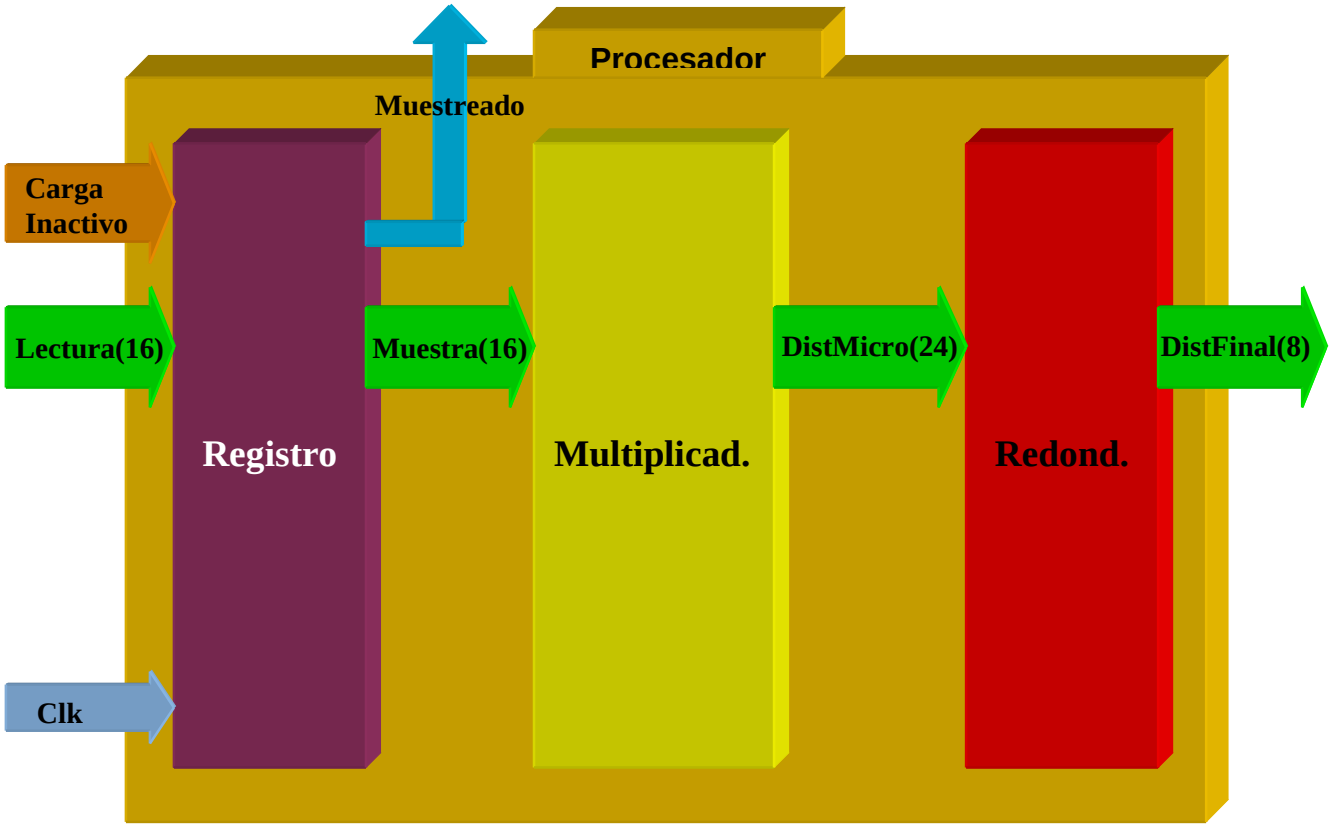


Figura 3.20: Diagrama de bloques VHDL del circuito Procesador

Con este diseño, que continua aplicando la filosofía Top-Down y se cumple toda la funcionalidad especificada. Dicho diseño es implementado con el siguiente código correspondiente al archivo *Procesador.vhd*.

A continuación se muestran las librerías y entidad de este circuito:

```
1  LIBRARY IEEE;                        --Librerias
2     USE ieee.std_logic_1164.all;
3     USE ieee.std_logic_arith.all;
4     USE ieee.std_logic_unsigned.all;
5
6
7  ENTITY Procesador IS                 --Entidad Procesador
8     GENERIC (NUMBITS_LECTURA, NUMBITS_MUESTRA : POSITIVE := 16; --Numero de Bits
9              NUMBITS_MICRO     : POSITIVE := 24;
10             NUMBITS_FINAL     : POSITIVE := 8);
11    PORT (Carga, Clk, Inactivo : in std_logic;                  --Entradas y Salidas
12          Lectura              : in std_logic_vector ((NUMBITS_LECTURA-1) DOWNTO 0);
13          Muestreado           : out std_logic;
14          DistFinal            : out std_logic_vector ((NUMBITS_FINAL-1) DOWNTO 0));
15 END Procesador;
16
```

En este diseño se utilizan todos los paquetes descritos en la introducción al VHDL para poder recurrir a ellos en cuanto a definición de tipos de datos, conversiones de tipos y operaciones se refiere.

Todos los datos genéricos de este código se utilizan para fijar las longitudes de los vectores que conforman la cadena de datos de la medida. Se observan los siguientes datos genéricos:

- *NUMBITS_LECTURA:* Este dato fija la longitud del vector de entrada llamado Lectura(16). Por lo que su valor para este diseño es 16.

- *NUMBITS_MUESTRA:* El contenido de este dato fija la longitud de un vector interno al procesador llamado Muestra(16). Por lo tanto su valor para este diseño será 16.

- *NUMBITS_MICRO:* Con este dato se ajusta la longitud de un vector interno llamado DistMicro(24), con lo cual este dato es igual a 24.

- *NUMBITS_FINAL:* Este dato fija la longitud del vector de salida llamado DistFinal(8), por lo que este valor será 8.

En la figura 3.17 se puede observar el bloque Procesador con sus entradas y salidas, que no son más que la expuestas en la entidad de este componente. De tal forma, en este diseño se declaran las siguientes entradas:

- *Carga:* Esta señal tiene como fin la actualización de la medida en un registro interno al Procesador. Cuando se pone a nivel alto, el Procesador actualiza el valor que tuviese guardado por el nuevo valor.

- *Inactivo:* Con esta señal se reseteará el anterior registro interno del que se habló cuando esta se establezca a nivel alto.

- *Lectura(16):* Este es el vector que permite la entrada del dato de la medida que proviene del Contador y cuya señal posee el mismo nombre. Este vector contiene la distancia en micrómetros.

Este componente, según el anterior código, también tendrá las salidas que se describen a continuación:

- *Muestreado:* Esta es una señal de sincronismo que permite al Secuenciador saber cuando el registro interno del Procesador ha guardado correctamente el dato.

- *DistFinal(8):* Este es el vector que permite la salida de datos del componente, ya convertidos a distancia y debidamente ajustada. Además este dato lleva implícito un código para indicar al Conversor si debe representarse como centímetros o como metros.

De tal modo queda introducido el encapsulado del circuito Procesador.

Este componente, por lo tanto, recibirá el tiempo de eco en microsegundos desde el Contador. El Procesador debe ser capaz de congelar este dato en un registro.

En la siguiente etapa se debe multiplicar este dato guardado por la velocidad del sonido en micrómetros por microsegundos obteniendo la distancia en micrómetros.

Por último, el dato debe ser redondeado y ajustado a centímetros o metros e indicarlo adecuadamente en el vector de salida.

El hecho de que en la última etapa se pase desde micrómetros a centímetros o metros aumenta la precisión y fiabilidad de la medida. Se estarán tomando medidas a una precisión muy por debajo de la que se mostrará finalmente y una pequeña desviación de una medida no influirá en el resultado final al filtrarla el redondeo.

Por otra parte, según lo comentado, se observa claramente como la función de este componente puede dividirse en tres procesos claramente diferenciados. Estos tres procesos conllevan tres subcircuitos que son: un Registro, un Multiplicador y un Redondeador. De este modo, el componente Procesador queda internamente estructurado según la figura 3.20 introducida.

A continuación se observa como todo lo especificado se plasma en el código correspondiente a la arquitectura Básica del Procesador:

```
18  ARCHITECTURE Basica OF Procesador IS      --Basica
19
20     COMPONENT Registro                        --Componente Registro
21        GENERIC (NUMBITS : POSITIVE := 16);
22        PORT (Carga, Clk, Inactivo : in std_logic;
23              Lectura              : in std_logic_vector ((NUMBITS-1) DOWNTO 0);
24              Muestreado           : out std_logic;
25              Muestra              : out std_logic_vector ((NUMBITS-1) DOWNTO 0));
26     END COMPONENT;
27
28     COMPONENT Multiplicador                   --Componente Multiplicador
29        GENERIC (NUMBITS_MUESTRA  : POSITIVE := 16;
30                 NUMBITS_MICRO    : POSITIVE := 24;
31                 VELOCIDAD_SONIDO : INTEGER  := 172);
32        PORT (Muestra   : in std_logic_vector ((NUMBITS_MUESTRA-1) DOWNTO 0);
33              DistMicro : out std_logic_vector ((NUMBITS_MICRO-1) DOWNTO 0));
34     END COMPONENT;
35
36     COMPONENT Redondeador                     --Componente Redondeador
37        GENERIC (NUMBITS_MICRO : POSITIVE := 24;
38                 NUMBITS_FINAL : POSITIVE := 8);
39        PORT (DistMicro : in std_logic_vector ((NUMBITS_MICRO-1) DOWNTO 0);
40              DistFinal : out std_logic_vector ((NUMBITS_FINAL-1) DOWNTO 0));
41     END COMPONENT;
42
43     SIGNAL Muestra : std_logic_vector ((NUMBITS_MUESTRA-1) DOWNTO 0);
44     SIGNAL DistMicro : std_logic_vector ((NUMBITS_MICRO-1) DOWNTO 0);
45
46     BEGIN
47        Reg: Registro PORT MAP                    --Instancia Registro
48           (Carga, Clk, Inactivo, Lectura, Muestreado, Muestra);
49
50        Multi: Multiplicador PORT MAP             --Instancia Multiplicador
51           (Muestra, DistMicro);
52
53        Redond: Redondeador PORT MAP              --Instancia Redondeador
54           (DistMicro, DistFinal);
55
56  END Basica;
```

Dado que este componente está compuesto por tres subcircuitos habrá que declarar los tres componentes que dan forma al Procesador. Estos son el Registro, el Multiplicador y el Redondeador; según el orden del proceso.

Posteriormente se declaran las dos señales que conectarán estos circuitos, estas son Muestra(16) y DistMicro(24), cuyos ajustes se mencionaron al comentar la entidad del Procesador.

Más adelante se comentará en detalle la implementación de cada uno de los tres componentes que conforman esta arquitectura. Sin embargo, se introducirá como adelanto y para explicar la parte descriptiva de la arquitectura, las conexiones entre los componentes.

Al Registro le entran todas las señales de entrada al Procesador, es decir, Carga, Inactivo y Lectura(16) y genera las señales Muestreado y Muestra(16).

En cuanto al multiplicador, tiene como entrada la señal Muestra(16) generada en el Registro y como salida DistMicro(24). Este circuito es meramente combinacional.

Por último, el Redondeador tiene como entrada el vector generado en el Multiplicador, es decir, DistMicro(24); y como salida el vector DistFinal(8).

3.4.3.A. Registro

Este componente se encuadra dentro del bloque Procesador según lo expuesto en la figura 3.20. Su función será la de conseguir memorizar una medida. Esto es necesario para poder mostrar dicha medida estática mientras se realiza otra medida y hasta que la misma finaliza.

En ese caso el Contador cambiará de valor para poder realizar otra medida pero el registro seguirá guardando la medida anterior y está será la mostrada en el display. Dicha medida será conservada mientras no el sistema no sea apagado o reseteado.

Las entradas, salidas y datos genéricos son los comentados en el bloque procesador; donde también se codifican en VHDL al ser declarado como componente el Registro.

No obstante, en este epígrafe se comentará en detalle todo el código de este *Registro* y su funcionamiento. De tal forma, el código VHDL que se muestra a continuación es el perteneciente a dicho componente, que esta contenido en el archivo *Registro.vhd*:

```
1  LIBRARY IEEE;                          --Librerias
2     USE ieee.std_logic_1164.all;
3     USE ieee.std_logic_arith.all;
4
5
6  ENTITY Registro IS                     --Entidad Registro
7     GENERIC (NUMBITS : POSITIVE := 16);       --Numero de Bits
8     PORT (Carga, Clk, Inactivo : in std_logic;   --Entradas y Salidas
9           Lectura              : in std_logic_vector ((NUMBITS-1) DOWNTO 0);
10          Muestreado           : out std_logic;
11          Muestra              : out std_logic_vector ((NUMBITS-1) DOWNTO 0));
12 END Registro;
13
```

Como se observa, en este código se declaran los paquetes std_logic_1164 y std_logic_arith de la librería IEEE.

A continuación se define, ya en la entidad, un dato genérico llamado NUMBITS. Este dato fijará la longitud de los vectores de entrada y salida, es decir, de Lectura(16) y Muestra(16).

Seguidamente se describen las entradas al componente, las cuales son idénticas al bloque Procesador: Carga, Inactivo y Lectura. En cuanto a las salidas, también se conoce su función ya que, Muestreado es una salida del

Procesador y Muestra(16) es el dato memorizado en el Registro que conecta con el Multiplicador.

En la arquitectura llamada *Medida*, que no posee parte declarativa, se puede apreciar el siguiente código:

```
14
15  ARCHITECTURE Medida OF Registro IS  --Medida
16
17     BEGIN
18        PROCESS (Clk, Inactivo, Carga)      --Proceso de Carga
19
20           BEGIN
21              IF Inactivo = '1' THEN                  --Estado de Inactividad
22                 Muestra ((NUMBITS-1) DOWNTO 0) <= "0000000000000000";
23                 Muestreado <= '0';
24
25              ELSIF Clk 'EVENT AND Clk = '0' THEN
26                 IF Carga = '1' THEN                  --Funcion de Carga
27                    Muestra ((NUMBITS-1) DOWNTO 0) <= Lectura ((NUMBITS-1) DOWNTO 0);
28                    Muestreado <= '1';                --Señal de Registro Cargado
29                 ELSE
30                    Muestreado <= '0';                --Reseteo Señal de Reg. Carg.
31                 END IF;
32
33              END IF;
34
35           END PROCESS;                  --Fin de Proceso de Carga
36
37  END Medida;
```

Lo primero que se observa al comenzar la parte descriptiva de la entidad es el proceso de carga, cuya lista de sensibilidad está compuesta por las señales Inactivo, Carga y Clk.

Este proceso tampoco tiene parte declarativa, por lo que comienza con la descriptiva. En esta parte se observa un bloque disyuntivo que hace que, cuando la señal Inactivo se encuentre a nivel alto, se ponga la salida Muestra(16) a cero y la señal Muestreado a nivel bajo (--Estado de Inactividad).

Si esta señal no está a nivel alto y además se da un flanco negativo de la señal de reloj Clk, se ejecuta otro código.

En este caso, si la señal Carga está a nivel alto, en la señal Muestra(16) se introduce Lectura(16). Cuando esto se ha producido y solo en ese momento, la señal Muestreado se pone a nivel alto (--Función de Carga)

Si la señal Carga no esta a nivel alto, la señal Muestreado se pondrá a nivel bajo (--Reseteo Señal de Reg. Carga).

Por lo tanto, la señal Muestreado permanecerá a nivel alto hasta que se ponga a nivel bajo la señal Carga. Esta señal, a su vez, se pondrá a nivel bajo cuando el Secuenciador reciba Muestreado a nivel alto y así la comunicación quedará cerrada.

Con esto se concluye la descripción de la arquitectura y queda totalmente implementada la funcionalidad del Registro

3.4.3.B. Multiplicador

Como su nombre indica, este componente incluido en el bloque Procesador, tiene como finalidad la multiplicación de la velocidad del sonido por el dato guardado en el Registro.

Como peculiaridades cabe mencionar que el dato de entrada de este circuito está expresado en microsegundos y la velocidad del sonido se expresa en micrómetros por microsegundo. Esto permitirá una mayor precisión en el resultado final del sistema.

Para resolver estas especificaciones según lo comentado se diseña el siguiente código VHDL que consta en el archivo *Multiplicador.vhd*:

```
1  LIBRARY IEEE;                        --Librerias
2     USE ieee.std_logic_1164.all;
3     USE ieee.std_logic_arith.all;
4     USE ieee.std_logic_unsigned.all;
5
6
7  ENTITY Multiplicador IS              --Entidad Multiplicador
8     GENERIC (NUMBITS_MUESTRA  : POSITIVE := 16;  --Numero de bits Entradas
9              NUMBITS_MICRO    : POSITIVE := 24;
10             VELOCIDAD_SONIDO : INTEGER  := 172); --Velocidad del Sonido / 2
11                                                  --Entradas y Salidas ->
12    PORT (Muestra   : in std_logic_vector ((NUMBITS_MUESTRA-1) DOWNTO 0);
13          DistMicro : out std_logic_vector ((NUMBITS_MICRO-1) DOWNTO 0));
14 END Multiplicador;
15
```

Lo primero que se aprecia es la declaración de librerías y paquetes. Debido a la operación de multiplicación, las conversiones de datos y demás funciones necesarias para la implementación de este módulo; en este componente se utilizan los tres paquetes descritos inicialmente de la librería IEEE.

En cuanto a la entidad, se observa la definición de tres datos genéricos que son los siguientes:

- *NUMBITS_MUESTRA:* Este dato contiene la longitud del vector Muestra(16), por lo que su valor será 16 para este sistema.
- *NUMBITS_MICRO:* Aquí se almacena la longitud del vector de salida DistMicro(24) y tiene un valor de 24.

- *VELOCIDAD_SONIDO:* En este dato se almacena la mitad de la velocidad del sonido para una atmósfera normal. Dicho valor es de, aproximadamente 172 micrómetros por microsegundo.

Las entradas y salidas de este componente ya han sido descritas anteriormente y son: la señal Muestra(16), que contiene el dato del tiempo de eco en microsegundos y proviene del Registro; y la señal DistMicro(24), que contiene el dato de la distancia en micrómetros.

En cuanto a la arquitectura de este circuito, se resume en una simple operación de alto nivel que se encuentra definida en los paquetes declarados. Esto se aprecia a continuación en dicha arquitectura llamada *Simple*:

```
16
17  ARCHITECTURE Simple OF Multiplicador IS   --Simple
18
19     SIGNAL Velocidad : STD_LOGIC_VECTOR (7 DOWNTO 0);
20
21     BEGIN
22        PROCESS (Muestra, Velocidad)          --Proceso de Multiplicacion
23
24           BEGIN
25              DistMicro <= Muestra * Velocidad;
26
27           END PROCESS;                       --Fin de Multiplicacion
28
29        Velocidad (7 DOWNTO 0) <= CONV_STD_LOGIC_VECTOR (VELOCIDAD_SONIDO, 8);  --Conv. V.S.
30
31  END Simple;
```

Para poder realizar la operación comentada anteriormente, se deben tener dos operadores de tipo std_logic, por este motivo se declara el vector Velocidad(8).

Este vector se asigna mediante una conversión del dato genérico VELOCIDAD_SONIDO al final de la arquitectura. Gracias a este proceso, la velocidad del sonido podrá ser ajustada fácilmente cambiando únicamente el dato genérico.

Por otra parte, se describe un proceso cuya lista de sensibilidad son los dos operadores de la multiplicación. En el interior del proceso se desarrolla la asignación a la salida DistMicro del resultado de la multiplicación.

Con esto finaliza la arquitectura y la descripción del circuito Multiplicador que, a pesar de su simplicidad, es la base del Medidor.

Sin embargo, antes de finalizar este epígrafe cabe mencionar un detalle utilizado en este componente. La velocidad del sonido para una atmósfera estándar, considerada como 344 m/s (o µm/µs), se divide entre dos para su uso. De hecho, solo se tiene la constante de la mitad de la velocidad del sonido.

Esto hace referencia a que para hallar la distancia real a un objeto se debe multiplicar tiempo de eco por velocidad del sonido (con coherencia de unidades) y dividir el resultado entre dos.

La última división entre dos es imprescindible puesto que el tiempo de eco que se cuenta es el tiempo en el que la onda de ultrasonidos recorre en ida y vuelta el camino desde la placa auxiliar al objeto interpuesto.

En este componente se divide entre dos la velocidad del sonido para economizar electrónica ahorrando un módulo destinado a dividir entre dos. La implementación de dicho módulo no sería excesivamente difícil y conllevaría un desplazamiento a la derecha de los bits del vector del resultado. Pero, en este caso se prefiere el sistema comentado.

En este punto sí quedan comentados el funcionamiento, codificación y detalles de este circuito llamado Multiplicador.

3.4.3.C. Redondeador

Este es el único componente de la ruta de datos implementada en el Procesador. La función de este circuito es fundamental para la precisión y fiabilidad de la medida dada por el sistema.

La función del *Redondeador* será la de ajustar la distancia dada en micrómetros a un dato redondeado y en precisión de centímetros o metros dependiendo de la magnitud. Además se debe de obtener un indicador de la precisión en la que se aporta el dato para que el Conversor sepa interpretar y mostrar el mismo en el display.

Como ya se aprecia a primera vista esta será una tarea muy poco trivial y de hecho, a parte de desarrollar bastante código, también supone mucha superficie en el circuito de la FPGA.

El código VHDL de este componente se guarda en el archivo *Redondeador.vhd*, que comienza con las librerías y entidad según se muestra a continuación:

```
1  LIBRARY IEEE;                        --Librerias
2     USE ieee.std_logic_1164.all;
3     USE ieee.std_logic_arith.all;
4     USE ieee.std_logic_unsigned.all;
5
6
7  ENTITY Redondeador IS                --Entidad Redondeador
8     GENERIC (NUMBITS_MICRO : POSITIVE := 24;  --Numero de Bits Entrada y Salida
9              NUMBITS_FINAL : POSITIVE := 8);  --Entradas y Salidas -->
10    PORT (DistMicro : in std_logic_vector ((NUMBITS_MICRO-1) DOWNTO 0);
11          DistFinal : out std_logic_vector ((NUMBITS_FINAL-1) DOWNTO 0));
12 END Redondeador;
13
```

Según lo que aparece en este fragmento de código, se utilizarán todos los paquetes descritos para la librería IEEE. Esto permitirá el posterior tratamiento de tipos de datos UNSIGNED y conversiones entre diversos tipos de datos.

Ya en la entidad se observan dos datos genéricos que fijan la longitud de los vectores de entrada y salida de este bloque. Estos son los siguientes:

- *NUMBITS_MICRO:* Este dato acota la longitud de la señal de entrada DistMicro(24) por lo que para este diseño posee el valor 24.

- *NUMBITS_FINAL:* La señal de salida DistFinal(8) es ajustada en longitud por este dato genérico que vale 8.

En cuanto a entradas y salidas del circuito, solamente posee dos que son la entrada de datos en micrómetros DistMicro(24) y la salida de datos codificados en centímetros o metros DistFinal(8).

Hasta aquí la entidad del Redondeador. Seguidamente comienza la arquitectura llamada *Comparador* debido a los bloques de comparación que incluye. Dicha arquitectura no posee parte declarativa, por lo que se comienza directamente con la descriptiva que se muestra a continuación:

```
14
15 ARCHITECTURE Comparador OF Redondeador IS --Comparador
16
17    BEGIN
18       PROCESS (DistMicro)                --Proceso de Redondeo por Comparacion
19
20          VARIABLE Valor   : INTEGER RANGE 0 TO 16777216; --Declaracion Variable Trabajo
21          VARIABLE mt_ncmt : STD_LOGIC;                  --Variables de Salida
22          VARIABLE Ajuste  : INTEGER RANGE 0 TO 100;
23
24          BEGIN
25             Valor := CONV_INTEGER (UNSIGNED(DistMicro)); --Inicializacion Variables
26             Ajuste := 0;
```

Lo primero que se describe en este código es un proceso cuya lista de sensibilidad es el vector de entrada DistMicro(24), es decir, que cada vez que cambie la entrada se producirá una conversión.

En la parte declarativa de este proceso se codifican tres variables que son las siguientes:

- *Valor:* Esta variable es de tipo INTEGER y puede contener cifras desde el 0 al 16.777.216, que es el máximo teórico que podría albergar un número binario de 24 bits como el vector que se le asignará DistMicro.

- *mt_ncmt:* La codificación de la representación del dato de salida como metros o centímetros se hará a través de la asignación al octavo bit de la señal de salida DistFinal(8) de esta variable de tipo std_logic. Si esta variable es '1' la distancia se expresa en metros y si es '0' en centímetros.

- *Ajuste:* Esta variable será, junto con Valor, una variable auxiliar para que, mediante bloques de comparación, se consiga ajustar y redondear correctamente la distancia. Concretamente, esta variable al final del proceso contendrá el valor de salida que se volcará en DistFinal(8) con mt_ncmt.

Tras la parte declarativa, comienza la descriptiva. Lo primero que se hace en esta parte son las inicializaciones de variables.

La primera de estas inicializaciones es la que vuelca la entrada DistMicro(24) a la variable de trabajo Valor. Para ello se debe convertir el tipo de esta entrada de STD_LOGIC a UNSIGNED. Al mismo tiempo, cuando el tipo ya es UNSIGNED, se convierte finalmente a INTEGER pudiendo ya ser asignado a la variable Valor.

La segunda de las inicializaciones es más sencilla y conlleva simplemente la asignación de un cero sobre la variable Ajuste.

Seguidamente empiezan los bloques de comparación. En primera instancia, en el nivel superior se encuentra el primer bloque de comparación que cuenta con tres instancias: la primera discrimina si la medida son centímetros, la segunda si son metros y la tercera cubre el resto de los casos que suponen un error.

A continuación se muestra el código VHDL de la primera instancia, cuyo fin será redondear y codificar la entrada, obteniendo la distancia final en centímetros en las variables Ajuste y mt_ncmt:

```
27
28              IF Valor <= 994999 THEN       --Ajuste a Nivel de Centimetros
29                 mt_ncmt := '0';              --Indica Centimetros
30
31                 CASE Valor IS                --Caso Decimetros
32                    WHEN 49999 TO 99999 => Ajuste := 0; --No mide nada < 5 cm.ERROR
33                    WHEN 100000 TO 199999 => Ajuste := 10;
34                                             Valor := Valor - 100000;
35                    WHEN 200000 TO 299999 => Ajuste := 20;
36                                             Valor := Valor - 200000;
37                    WHEN 300000 TO 399999 => Ajuste := 30;
38                                             Valor := Valor - 300000;
39                    WHEN 400000 TO 499999 => Ajuste := 40;
40                                             Valor := Valor - 400000;
41                    WHEN 500000 TO 599999 => Ajuste := 50;
42                                             Valor := Valor - 500000;
43                    WHEN 600000 TO 699999 => Ajuste := 60;
44                                             Valor := Valor - 600000;
45                    WHEN 700000 TO 799999 => Ajuste := 70;
46                                             Valor := Valor - 700000;
47                    WHEN 800000 TO 899999 => Ajuste := 80;
48                                             Valor := Valor - 800000;
49                    WHEN 900000 TO 994999 => Ajuste := 90;
50                                             Valor := Valor - 900000;
51                    WHEN OTHERS => Valor := 100000;  --ERROR
52                 END CASE;
53
54                 CASE Valor IS                --Caso Centimetros y Redondeador
55                    WHEN 0 TO 4999 => Ajuste := Ajuste + 0;
56                    WHEN 5000 TO 14999 => Ajuste := Ajuste + 1;
57                    WHEN 15000 TO 24999 => Ajuste := Ajuste + 2;
58                    WHEN 25000 TO 34999 => Ajuste := Ajuste + 3;
59                    WHEN 35000 TO 44999 => Ajuste := Ajuste + 4;
60                    WHEN 45000 TO 54999 => Ajuste := Ajuste + 5;
61                    WHEN 55000 TO 64999 => Ajuste := Ajuste + 6;
62                    WHEN 65000 TO 74999 => Ajuste := Ajuste + 7;
63                    WHEN 75000 TO 84999 => Ajuste := Ajuste + 8;
64                    WHEN 85000 TO 94999 => Ajuste := Ajuste + 9;
65                    WHEN 95000 TO 99999 => Ajuste := Ajuste + 10;
66                    WHEN OTHERS => Ajuste := 100;    --ERROR
67                 END CASE;
```

La primera sentencia que se observa en este código es la primera instancia del primer nivel de comparación que discierne entre si la distancia se debe expresar en metros o en centímetros (--Ajuste a Nivel de Centímetros).

Esta sentencia es de tipo IF y hace que cuando la variable Valor sea menor o igual a 994.999 micrómetros se entre en el nivel inferior de comparación para redondear la cifra en cuestión.

Según las reglas de redondeo científicas, al ajustar a una cierta precisión, cuando se alcanza o rebasa la mitad de la mínima precisión a redondear se redondeará al alza y en caso contrario a la baja. Esta regla de redondeo se seguirá siempre en adelante.

Es por esto que al ajustar a centímetros, si el dato de entrada en Valor supera los 99,4999 cm. (máxima precisión a micrómetros) y llega a los 99,5 cm. se debe redondear a 1 metro y se debe entrar en la segunda instancia al mayor nivel de comparación.

Lo primero que se hace al entrar en este bloque es asignar el valor ‘0’ a la variable mt_ncmt. Al encontrarse dentro de este bloque de comparación se puede afirmar que la distancia se puede expresar con dos cifras en centímetros.

Este es el discriminante entre la representación en metros centímetros ya que en el display siempre se representan dos dígitos y un símbolo que identifica la precisión. Por lo tanto, por encima de 99 centímetros no se puede representar ninguna distancia en centímetros en dos dígitos decimales y se pasará a medir en metros.

Seguidamente se entra en el nivel inferior de comparación donde se encuentra una sentencia de tipo CASE. Con esta sentencia se discriminará en qué decímetro se encuentra la distancia medida (--Caso Decímetros).

Como los decímetros no son la mínima precisión a redondear, los intervalos de comparación van desde X00.000 a X99.999 micrómetros, es decir, desde X0 a X9 centímetros.

Cada sentencia de comparación en rangos realiza dos funciones. La primera es asignar el valor de las decímetros a la variable Ajuste en centímetros. La segunda es restar a la variable Valor el mismo número de decímetros que hay en Ajuste expresadas en micrómetros.

Ambas acciones dejan preparadas las variables para que sean correctamente utilizadas en el nivel inmediatamente inferior de comparación. La primera acción prefija los decímetros para determinar seguidamente los centímetros y sumárselos a esa variable Ajuste. La segunda acción se encarga de eliminar los decímetros de la variable Valor para poder comparar solo centímetros en el siguiente nivel de comparación.

El nivel inferior de comparación es otra sentencia CASE que se comentará, donde se recibe las variables Ajuste y Valor con cifras prefijadas para este bloque que se comenta a continuación (--Caso Centímetros y Redondeador).

Este bloque de comparación es el que redondea a centímetros con el proceso que se explicó anteriormente. Por lo tanto, los rangos de comparación irán desde X5000 hasta (X+1)4999 micrómetros. Esto asegura el correcto redondeo de la cifra.

Para cada comparación, en este caso, solamente se realiza una acción. Dicha acción es la suma de los centímetros hallados en función de la

comparación, respetando el anterior valor asignado a Ajuste, y guardándolo todo junto en esta variable.

Existen dos excepciones a lo comentado:

- En el bloque de comparación en decímetros, la mínima distancia para los 0 decímetros son 5 centímetros. Por debajo de esta cifra el sistema mostrará Erro en el display ya que esta medida puede venir de un acople entre emisor y receptor de ultrasonidos sin producirse el fenómeno del eco. Esta limitación se muestra en especificaciones (No mide nada < 5 cm.ERROR).

- En el bloque de comparación de centímetros, los intervalos de comparación mínimo y máximo cubren la mitad de rango que el resto. Es decir, que el redondeo de 0 centímetros cubre desde 0 a 4.999 micrómetros y el redondeo a un nuevo decímetro cubre desde 95.000 a 99.999 micrómetros.

Por otra parte, solo queda comentar el sistema para provocar que el Convertidor muestre en el display el mensaje Erro.

En el bloque de decímetros se tiene una sentencia muy saludable en el caso del CASE descrita como WHEN OTHERS. Cuando la variable Valor no se encuadra en ninguno de los rangos de comparación se ejecutará esta sentencia. Esto provoca que a Valor se le asigne el dato 100.000 que forzará el error. Teóricamente esta sentencia solamente se ejecutará cuando la distancia medida es menor de 5 centímetros aunque también puede detectar otro tipo de errores más extraños (--ERROR).

En el bloque centímetros se encuentra la misma sentencia WHEN OTHERS. Si la variable Valor no se encuadra en ningún rango se ejecuta esta sentencia.

Esto puede suceder por dos motivos: un error descontrolado en el circuito que, por rangos no debería suceder; o bien, el más probable, que sea un error forzado del caso anterior porque la distancia medida es menor a 5 cm. Al asignar 100.000 a la variable Valor, no se entrará en ningún rango de este bloque ya que la mayor cifra lógica es 99.999 micrómetros.

En esta sentencia se asignará a Ajuste el valor de 100. Como ya se ha comentado, el Conversor siempre mostrará dos dígitos decimales y un símbolo que los identifique como metros o centímetros. Por lo tanto, si se pasa un valor de 100 a este componente, el mismo mostrará en el display el mensaje Erro.

De este modo se ha gestionado un sistema para poder detectar cualquier tipo de error en la medida y mostrarlo en el display para que el usuario sea consciente de que está sucediendo alguna anomalía.

Por otra parte, la segunda instancia del nivel superior de comparación es la que determina los metros si así lo indica la medida. Este código, muy semejante al anterior, se muestra a continuación:

```
68
69              ELSIF Valor <= 6049999 THEN   --Ajuste a Nivel de Metros
70                 mt_ncmt := '1';              --Indica Metros
71
72                 CASE Valor IS                --Caso Metros
73                    WHEN 0 TO 999999 => Ajuste := 0;
74                    WHEN 1000000 TO 1999999 => Ajuste := 10;
75                                               Valor := Valor - 1000000;
76                    WHEN 2000000 TO 2999999 => Ajuste := 20;
77                                               Valor := Valor - 2000000;
78                    WHEN 3000000 TO 3999999 => Ajuste := 30;
79                                               Valor := Valor - 3000000;
80                    WHEN 4000000 TO 4999999 => Ajuste := 40;
81                                               Valor := Valor - 4000000;
82                    WHEN 5000000 TO 5999999 => Ajuste := 50;
83                                               Valor := Valor - 5000000;
84                    WHEN 6000000 TO 6049999 => Ajuste := 60;
85                                               Valor := Valor - 6000000;
86                    WHEN OTHERS => Valor := 1000000; --No mide nada > 6 m.ERROR
87                 END CASE;
88
89                 CASE Valor IS                --Caso Decimetros y Redondeador
90                    WHEN 0 TO 49999 => Ajuste := Ajuste + 0;
91                    WHEN 50000 TO 149999 => Ajuste := Ajuste + 1;
92                    WHEN 150000 TO 249999 => Ajuste := Ajuste + 2;
93                    WHEN 250000 TO 349999 => Ajuste := Ajuste + 3;
94                    WHEN 350000 TO 449999 => Ajuste := Ajuste + 4;
95                    WHEN 450000 TO 549999 => Ajuste := Ajuste + 5;
96                    WHEN 550000 TO 649999 => Ajuste := Ajuste + 6;
97                    WHEN 650000 TO 749999 => Ajuste := Ajuste + 7;
98                    WHEN 750000 TO 849999 => Ajuste := Ajuste + 8;
99                    WHEN 850000 TO 949999 => Ajuste := Ajuste + 9;
100                   WHEN 950000 TO 999999 => Ajuste := Ajuste + 10;
101                   WHEN OTHERS => Ajuste := 100;    --ERROR
102                END CASE;
```

Como se observa se entrará en este bloque si, a parte de que en Valor no se encuentre una cifra que se pueda representar en centímetros, tampoco se encuentra una cifra que al redondear sea mayor a 6 metros. Se supone que este es el rango máximo de medida de sensores y lo será también de este medidor, por lo que un dato por encima de esta cifra se tomará como un error en la medida (--Ajuste a Nivel de Metros).

Si finalmente se entra en este bloque, lo primero que se hace es asignar a la variable mt_cmt el valor ‘1’ que hará que el bloque conversor disponga un identificador de metros en el display.

Seguidamente se ejecuta una sentencia CASE que supone un nivel inferior de comparación para fijar cual es exactamente el número de metros a representar. Esta sentencia es semejante a la de decímetros del código anterior. Sin embargo, solo se difiere en el caso de error por debajo de 5 centímetros y, además, habrá que añadir a todos los límites un 0 o 9 dependiendo si inicia o finaliza el rango, respectivamente.

En este código el caso de error se gestiona por encima de los 6.049.999 micrómetros; valor que se redondearía por encima de los 6 metros. No es posible que esto suceda según los bloques expuestos, pero es una buena práctica tenerlo en cuenta. Pero en este caso, en la sentencia WHEN OTHERS, Valor será igual a 1.000.000 de micrómetros ya que en el siguiente bloque no habrá nada por encima de esta cifra.

El nivel inmediatamente inferior será el siguiente bloque CASE de comparación y redondeo de decímetros. Al igual que lo sucedido con el bloque de metros, este bloque es semejante al de centímetros del anterior código comentado. Esta vez la gestión de error será idéntica asumiendo Ajuste la cifra de 100; pero habrá que añadir también un 0 o 9 a los límites de los rangos de comparación.

Como ya se ha comentado, cuando el Conversor se encuentre con el valor 100 mostrará en el display el mensaje Erro.

Por último, queda comentar la tercera instancia del mayor nivel de comparación y cerrar la estructura del proceso así como la arquitectura. Esto se observa en el siguiente código que se muestra:

```
103
104             ELSE
105                Ajuste := 100;             --ERROR
106                mt_ncmt := '1';            --Indica Metros
107
108             END IF;
109
110             DistFinal (6 DOWNTO 0) <= CONV_STD_LOGIC_VECTOR (Ajuste, 7);
111             DistFinal (NUMBITS_FINAL - 1) <= mt_ncmt;
112
113          END PROCESS;                      --Fin de Redondeo
114
115  END Comparador;
```

En la última instancia de comparación, solamente se gestionará el caso de error de que la cifra de entrada no se encuadre entre los límites que van de 0 centímetros a 6 metros.

En este caso, es posible que el Contador haya llegado a desbordarse porque el receptor de ultrasonidos no ha recibido el eco. Entonces, el Contador pone a nivel alto la señal NoFin. De tal forma, el Secuenciador ordenada guardar en el registro del Procesador la cifra del máximo fijado en el Contador. Al multiplicar esa cifra por la mitad de la velocidad del sonido, la distancia obtenida será mayor a 6 metros.

El Redondeador interpreta que esto no es posible debido a las limitaciones del emisor y receptor de ultrasonidos. Y de esta forma, fija el valor de la variable Ajuste a 100 para que el Conversor muestre posteriormente el mensaje Erro en el display.

Seguidamente a esta instancia se cierra el bloque de comparación de mayor nivel y se realizan dos asignaciones. Al bit de mayor peso de la señal de salida DistFinal, es decir a DistFinal[8], se le asigna la variable mt_cmt. Al resto del vector de salida se le asigna la conversión a STD_LOGIC_VECTOR de la variable Ajuste.

Así se cierra el proceso que no se volverá a ejecutar hasta que la señal DistMicro(24) cambie, es decir, hasta que el Registro no se actualice con un dato distinto.

También se cierra la arquitectura de este componente al cumplirse ya toda la funcionalidad que se esperaba de este circuito.

Por último y para completar la comprensión de los bloques de comparación de este circuito, se expondrá un ejemplo de medición.

Si se emite una onda de ultrasonidos a unos 18 centímetros de un obstáculo, hasta que se reciba su eco pasarán 1.046 microsegundos, desplazándose a 344 m/s. Esta será por lo tanto la cifra que contendrá el Contador cuando el Secuenciador lo detenga.

La siguiente acción del Secuenciador será almacenar dicho dato en el Registro del Procesador mediante la señal Carga. Una vez que este dato sea guardado, el Secuenciador recibirá del registro la señal Muestreado y pasará a otro estado.

Pero lo relevante para este ejemplo es que el Multiplicador hará lo propio hallando el producto de estos 1.046 microsegundos con los 172 micrómetros por microsegundo de la velocidad del sonido. El resultado será la distancia de 179.912 micrómetros.

Cuando este dato llega al Redondeador, la primera instancia del bloque superior de comparación detectará que esta cifra, ya en la variable Valor, es menor que 994.999 micrómetros y se entrará en el bloque de muestreo en centímetros poniendo la variable mt_ncmt a ‘0’.

En la primera sentencia CASE se detectará que la variable Valor se encuentra acotada entre 100.000 y 199.999 micrómetros. Al entrar en esta sentencia se asignará a la variable Ajuste el valor de 10 centímetros. Seguidamente a la variable Valor se le restará la misma cifra en micrómetros, es decir, 179.912 – 100.000; obteniendo el resultado de 79.912 centímetros.

Con estos valores se afronta la siguiente sentencia CASE. El rango de Valor ajustado en este bloque es el que va de 75.000 a 84.999 micrómetros. Esta sentencia suma al valor que haya en Ajuste 8 centímetros.

Por lo tanto el resultado final de ajuste será de 18 y su representación según mt_ncmt será en centímetros, coincidiendo con la medida real. Estos datos se vuelcan sobre la señal de salida DistFinal(8) para ser representados adecuadamente en el componente Conversor que se describirá a continuación.

3.4.4. Conversor

Según lo mencionado, este es el último componente de la cadena de datos del medidor. Dicho circuito se encarga de realizar todo el tratamiento de datos con el fin de mostrar el resultado de las medidas en el display de la placa de desarrollo Spartan III.

Este tratamiento o conversión de datos se realiza según las especificaciones de diseño del Medidor de Distancia así como las especificaciones de utilización de la placa de desarrollo Spartan III.

Por lo tanto, el *Conversor* debe ser capaz de separar en dos dígitos en BCD la medida que recibe del Procesador e interpretar el código para conocer si la precisión de dicha medida es de metros o centímetros. Una vez tenga aislados los cuatro caracteres a representar en cada uno de los dígitos del display, debe de multiplexarlos temporalmente.

Para abordar la solución a los requerimientos mencionados anteriormente sobre este circuito, se divide su arquitectura en dos hitos de menor magnitud según la filosofía Top-Down.

De esta forma dentro del circuito Conversor habrá dos bloques funcionales. El primero tratará DistFinal(8) para obtener una secuencia de cuatro números BCD multiplexados temporalmente según una señal que saldrá directamente a AN(4). El segundo de estos bloques es un conversor de BCD a código 7 segmentos que sale a SieteSeg(8) y que además gestiona mensajes de error y caracteres como los de las precisiones en metros y centímetros.

Cabe mencionar que el primer bloque también calcula el refresco mínimo o que el segundo de estos bloques necesita conocer cual es el dígito que está activo en cada momento en la multiplexión para la gestión del punto, caracteres de precisión o mensaje Erro.

De tal modo el circuito Conversor se puede esquematizar según la siguiente figura 3.21:

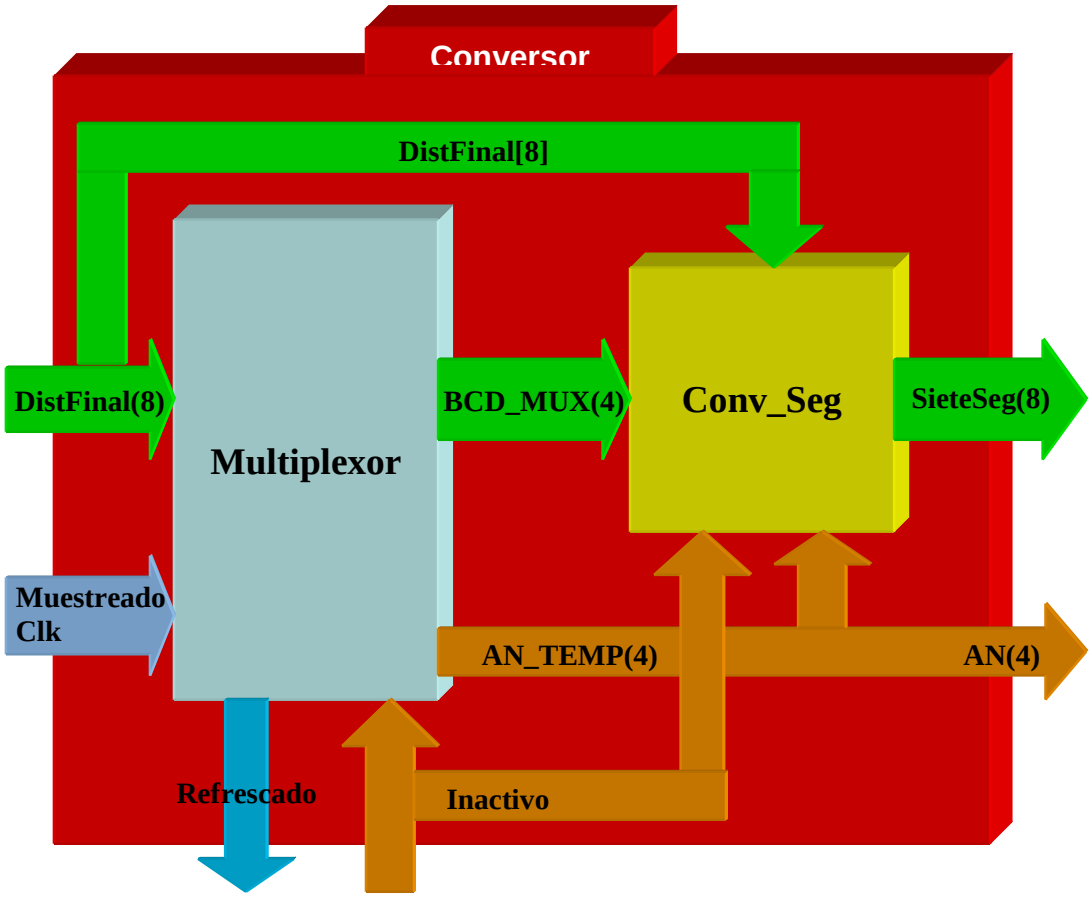


Figura 3.21: Diagrama de bloques VHDL del circuito Conversor

Estos dos bloques se podrían haber sintetizado en solo uno junto con el Redondeador del Procesador. Sin embargo, se ha optado por un sistema con componentes mucho más acotados y específicos. Esta elección se basa en el aprovechamiento de una de las ventajas más notables del VHDL, la reutilización de código.

Todo esto se desarrollará seguidamente explicando el código VHDL resultante que se encuentra en el archivo *Conversor.vhd*.

A continuación se muestran las librerías utilizadas y la entidad correspondiente a este circuito:

```
1  LIBRARY IEEE;                        --Librerias
2     USE ieee.std_logic_1164.all;
3     USE ieee.std_logic_arith.all;
4     USE ieee.std_logic_unsigned.all;
5
6
7  ENTITY Conversor IS                  --Entidad Conversor
8     GENERIC (NBITS : POSITIVE := 8); --Numero de bits entrada
9     PORT (DistFinal     : in std_logic_vector ((NBITS-1) DOWNTO 0);   --Entradas y Salidas
10          Clk, Inactivo : in std_logic;
11          Muestreado    : in std_logic;
12          Refrescado    : out std_logic;
13          AN            : out std_logic_vector (3 DOWNTO 0);
14          SieteSeg      : out std_logic_vector (7 DOWNTO 0));
15 END Conversor;
16
```

Lo primero que se aprecia en este código es la declaración de librerías y paquetes. De la librería IEEE se declaran los tres paquetes usados habitualmente para declarar tipos de datos como el UNSIGNED, para conversiones de tipos de datos o para operaciones. Estos paquetes son el std_logic_1164, el std_logic_arith, y el std_logic_unsigned.

Seguidamente comienza la entidad del circuito. En este caso solamente existe un dato genérico llamado NBITS para el vector de entrada DistFinal(8). Este dato genérico tendrá para este diseño un valor de 8 coincidiendo para que la longitud del vector de entrada coincida con el de salida del Procesador.

En cuanto a entradas y salidas de este circuito se refiere, se pueden observar las siguientes entradas al mismo:

- *DistFinal(8):* Como ya se ha comentado este es el vector por el cual se reciben los datos del Procesador. El bit 8º contiene la información acerca de si el dato a representar tiene precisión de metros o centímetros y el resto de bits contienen el dato a representar.

- *Inactivo:* Esta señal, como su nombre indica, provoca la inactividad del Conversor. Esto supone que no se representa nada en el display de salida de medidas.

- *Muestreado:* Cuando se recibe esta señal a nivel alto se comienza el cálculo del refresco mínimo establecido para el display. Esto es debido a que en ese momento, la medida a representar es una nueva medida actualizada.

En cuanto a las salidas del circuito, muy relacionadas con la representación de las medidas, se pueden observar las siguientes:

- *Refrescado:* Esta señal se pondrá a nivel alto cuando se produzca el refresco mínimo fijado para este sistema. Dicho refresco se tendrá en cuenta a partir de que cambie la medida a representar según la señal Muestreado.

- *AN(4):* Este vector es el que lleva las señales de los ánodos comunes de cada dígito. Mediante la rotación virtual de un '0' entre sus 4 posiciones junto con la sincronización del dato a representar hace que sea posible la multiplexión temporal y el uso del display de la placa.

- *SieteSeg(8):* Este es el vector de datos de salida. Sus posiciones, contienen desde la 8ª a la 2ª, un carácter en código 7 segmentos; y en la 1ª posición el punto decimal de cada dígito. Dicho vector se conecta a cátodos de los dígitos del display. Gracias a su cambio y sincronización con AN(4) se pueden representar medidas en dicho display de la placa.

Esta es la descripción del encapsulado de este circuito. Dichas señales son las que dan la funcionalidad descrita a este circuito y al Medidor para conseguir las especificaciones totales del sistema.

En el código VHDL mostrado a continuación se muestra la arquitectura *MultConv* que hace referencia a la figura 3.21:

```
17
18  ARCHITECTURE MultConv OF Conversor IS  --Multiplexor-Conversor
19
20     COMPONENT Multiplexor                  --Componente Multiplexor a BCD
21        GENERIC (NBITS : POSITIVE := 7);
22        PORT (DistFinal     : in std_logic_vector ((NBITS-1) DOWNTO 0);
23              Clk, Inactivo : in std_logic;
24              Muestreado    : in std_logic;
25              Refrescado    : out std_logic;
26              BCD_MUX, AN   : out std_logic_vector (3 DOWNTO 0));
27     END COMPONENT;
28
29     COMPONENT Conv_seg                     --Componente Conversor BCD-7 Segmentos
30        PORT (BCD_MUX          : in std_logic_vector(3 DOWNTO 0);
31              AN               : in std_logic_vector(3 DOWNTO 0);
32              mt_ncmt, Inactivo : in std_logic;
33              SieteSeg         : out std_logic_vector(7 DOWNTO 0));
34     END COMPONENT;
35
36     SIGNAL BCD_MUX : std_logic_vector (3 DOWNTO 0); --Señales Internas
37     SIGNAL AN_TEMP : std_logic_vector (3 DOWNTO 0);
38     SIGNAL mt_ncmt : std_logic;
39
40     BEGIN
41        Bloque1: Multiplexor PORT MAP            --Instancia Multiplexor
42              (DistFinal (6 DOWNTO 0), Clk, Inactivo, Muestreado, Refrescado, BCD_MUX, AN_TEMP);
43
44        Bloque2: Conv_seg PORT MAP               --Instancia Conversor
45              (BCD_MUX, AN_TEMP, mt_ncmt, Inactivo, SieteSeg);
46
47        mt_ncmt <= DistFinal (7);                --Asignacion de Señales Intermedias
48        AN <= AN_TEMP;
49
50  END MultConv;
```

Lo primero que se hace en esta arquitectura es declarar los bloques anteriormente mencionados como componentes del Conversor. Así se declaran los componentes Multiplexor y Conv_Seg.

Antes de terminar la parte declarativa de la entidad se establecen tres señales internas que se aprecian en la figura 3.21. Estas son: BCD_MUX(4) para la unión de datos entre los dos bloques, AN_TEMP(4) para la entrada de dato multiplexado a Conv_Seg, y mt_ncmt para la entrada de precisión a Conv_Seg.

En la parte descriptiva de la arquitectura se asignan las señales a cada componente según todo lo comentado. Cabe mencionar la asignación del 8º bit del vector de entrada DistFinal(8) a la señal interna mt_ncmt para la asignación de precisión al bloque Conv_Seg así como la asignación de AN_TEMP a AN para la salida del circuito.

Con esta descripción queda definido el contenido del Conversor y se pasa a definir cual es el código interno de los bloques Multiplexor y Conv_Seg que componen al Conversor.

3.4.4.A. Multiplexor

Este circuito forma parte del componente del Medidor llamado Conversor. Gracias a él, la funcionalidad del Conversor puede ser resuelta en dos bloques más simples, altamente modulares y reutilizables.

Este bloque es el más complejo de los dos en los que se divide el Conversor. Por ello es el generador de la multiplexión temporal del mismo así como el gestor del refresco mínimo.

Su función primaria para poder multiplexar los datos de las medidas es la conversión de código binario a código BCD. Posteriormente se mulltiplexan dichos dígitos ya en código BCD.

La consecución de dichos hitos será también desgranada en varios componentes cada uno con una función muy acotada en su ámbito y estándar, identificando en cada caso la necesidad. Todo este diseño queda esquematizado, con las señales internas asignadas así como otros detalles, según la siguiente figura 3.22:

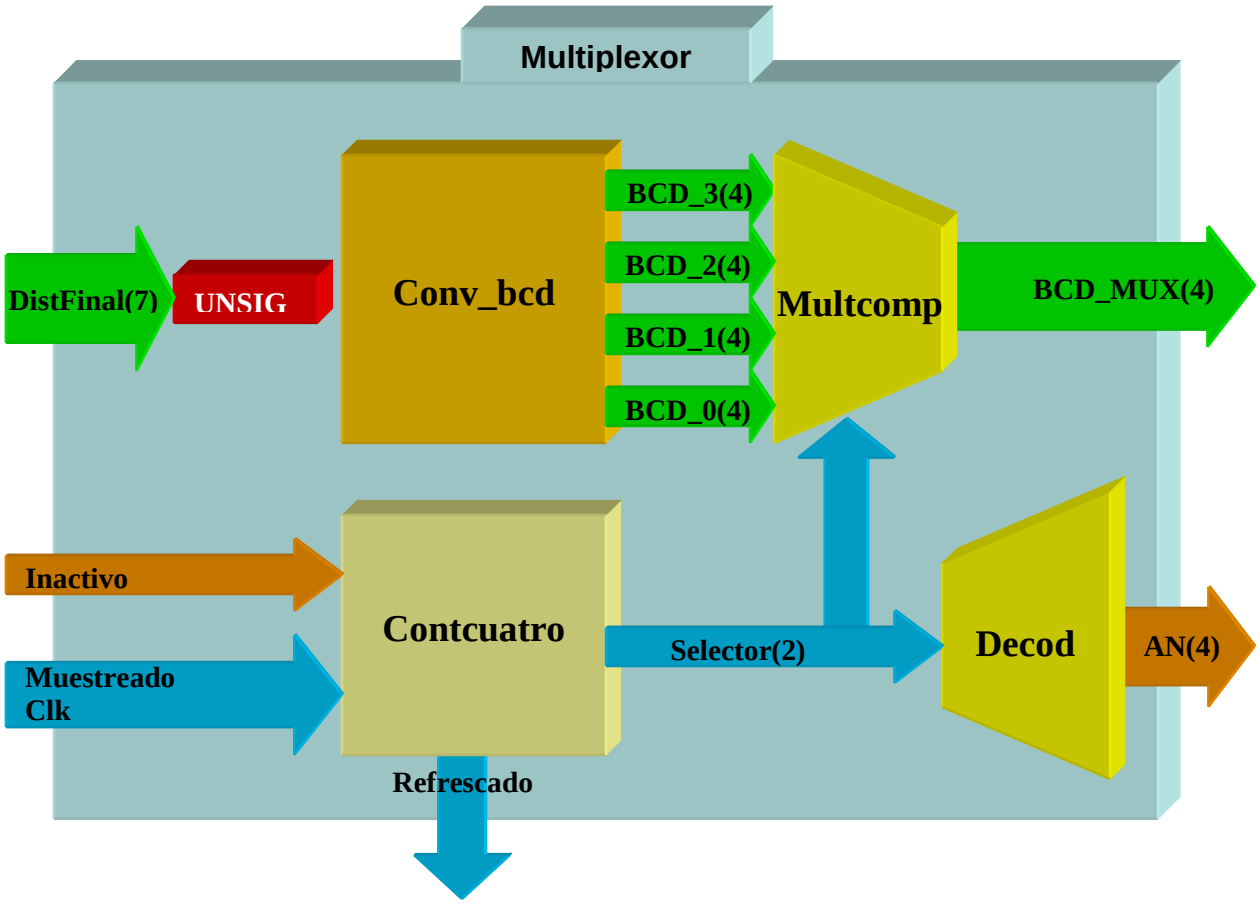


Figura 3.22: Diagrama de bloques VHDL del circuito Multiplexor

Según la figura, internamente el Multiplexor está estructurado según cuatro bloques funcionales menos complejos que el conjunto de todos ellos funcionando en común que se implementa finalmente. De tal modo, estos bloques serán los siguientes:

- *Conv_bcd:* Este bloque obtiene, a partir de DistFinal(7), cuatro códigos BCD bien identificados para que cada uno se le asigne posteriormente a un dígito concreto del display de la placa.

- *Contcuatro:* Este bloque no es más que un contador de módulo cuatro para generar la multiplexión temporal de los datos en el display. Además se hace cargo de la gestión del refresco ya que conoce perfectamente el número de refrescos realizados.

- *Multcomp:* A partir del número generado por el bloque Contcuatro, este componente consigue obtener un único código BCD a la salida del Multiplexor a partir de cuatro códigos de entrada del Conv_bcd.

- *Decod:* En concordancia con el anterior dato BCD que saca Multcomp, este componente genera el vector AN(4) con todos sus bits a '1', excepto el correspondiente al dígito del display en el que se tiene que representar ese código BCD; que se encontrará a '0'.

Todo este código está contenido en el archivo *Multiplexor.vhd*. Las librerías utilizadas en este bloque así como su entidad se definen según el código VHDL que se muestra a continuación:

```
1  LIBRARY IEEE;                        --Librerias
2     USE ieee.std_logic_1164.all;
3     USE ieee.std_logic_arith.all;
4     USE ieee.std_logic_unsigned.all;
5
6
7  ENTITY Multiplexor IS                --Entidad Multiplexor
8     GENERIC (NBITS : POSITIVE := 7); --Numero de bits entrada
9     PORT (DistFinal     : in std_logic_vector ((NBITS-1) DOWNTO 0);   --Entradas y Salidas
10          Clk, Inactivo : in std_logic;
11          Muestreado    : in std_logic;
12          Refrescado    : out std_logic;
13          BCD_MUX, AN   : out std_logic_vector (3 DOWNTO 0));
14 END Multiplexor;
15
```

La librería usada para este circuito es la IEEE que contiene los tres paquetes descritos y usados habitualmente en este proyecto para la declaración de tipos de objetos, conversiones y operaciones definidas.

Al igual que el componente Conversor, este circuito solamente tiene un dato genérico que es NBITS fijado a siete para concretar la longitud de la entrada del vector DistFinal(8). Hay que tener en cuenta que para este bloque no es necesario conocer la precisión de la medida; por lo tanto, en este caso solo entran los bits de este vector que van desde el 7º al 1º en DistFinal(7), siendo este dato genérico igual a siete.

En cuanto a los puertos de este circuito, asignados desde la arquitectura de la entidad Conversor, se definen las siguientes salidas:

- *DistFinal(7):* Este es el vector asignado mediante el dato genérico NBITS del que ya se ha hablado. Su contenido proviene del Procesador y, en este caso, contiene únicamente la medida en rango de 0 a 100.

- *Inactivo:* Esta señal se propaga a los dos bloques que componen el Conversor para realizar la acción mencionada, es decir, que se neutralice totalmente el display de la placa Spartan III.

- *Muestreado:* Como se introdujo anteriormente, esta es la señal a partir de la cual se calcula el refresco mínimo que se ha de producir

antes de comenzar una medición desde que el display cambia su medida.

Las salidas que le corresponden a este bloque son las siguientes:

- *Refrescado:* Esta señal de salida indica a nivel alto cuando se ha producido el refresco mínimo de un dato en el display según lo comentado anteriormente.

- *BCD_MUX(4):* Este es el vector de salida de datos en código BCD multiplexados hacia el bloque Conv_Seg. Como es de esperar, dadas las características del código BCD, la longitud de este bloque es 4 bits.

- *AN(4):* Este es el vector que marca el dígito que permanece activo en cada momento, coincidiendo con el dígito BCD que se saque del bloque.

Haciendo referencia a la figura 3.22, se pueden apreciar los cuatro módulos necesarios que se instancian en la arquitectura del Multiplexor como componentes y que se comentarán más adelante. También se observan todas las señales internas que se utilizan para la conexión de los cuatro componentes mencionados.

Todo esto es implementado en el código VHDL correspondiente a la arquitectura *Temporal* de la entidad Multiplexor. Este código se muestra a continuación:

```
16
17  ARCHITECTURE Temporal OF Multiplexor IS   --Temporal
18
19     COMPONENT Conv_bcd                        --Componente Coversor a BCD
20        GENERIC (NUMBITS  : POSITIVE := 7);
21        PORT (DistFinal : in UNSIGNED ((NUMBITS-1) DOWNTO 0);
22              BCD_3, BCD_2, BCD_1, BCD_0 : out std_logic_vector (3 DOWNTO 0));
23     END COMPONENT;
24
25     COMPONENT Contcuatro                      --Componente Contador a 4
26        GENERIC (DIV_FREQ : POSITIVE   := 5000;
27                 VISUALIZAR : POSITIVE := 17000000);
28        PORT (Clk, Inactivo, Muestreado : in std_logic;
29              Refrescado                : out std_logic := '1';
30              Selector                  : out std_logic_vector (1 DOWNTO 0));
31     END COMPONENT;
32
33     COMPONENT Multcomp                        --Componente Mux 4-1 4 bits
34     PORT (Selector                  : in std_logic_vector (1 DOWNTO 0);
35           BCD_3, BCD_2, BCD_1, BCD_0 : in std_logic_vector (3 DOWNTO 0);
36           BCD_MUX                    : out std_logic_vector (3 DOWNTO 0));
37     END COMPONENT;
38
39     COMPONENT Decod                           --Componente Decodificador 2-4
40        PORT (Selector : in std_logic_vector (1 DOWNTO 0);
41              AN       : out std_logic_vector (3 DOWNTO 0));
42     END COMPONENT;
43
44     SIGNAL BCD_3, BCD_2, BCD_1, BCD_0 : std_logic_vector (3 DOWNTO 0); --Señales Internas
45     SIGNAL Selector : std_logic_vector (1 DOWNTO 0);
46     SIGNAL Conversion : UNSIGNED ((NBITS-1) DOWNTO 0);
```

Como es de esperar, en la parte declarativa de esta arquitectura se realizan las instancias de los componentes a utilizar. Como ya se ha comentado estos circuitos serán cuatro, y sus nombres son *Conv_bcd, Contcuatro, Multcomp y Decod*.

En esta parte también se declaran todas las señales internas que se muestran en la figura 3.22; estas son: BCD_3, BCD_2, BCD_1, BCD_0, Selector(2) y Conversión(7). La señal que menos claramente se percibe es el vector tipo UNSIGNED Conversion(7) que se utiliza para asignar el vector de entrada DistFinal(7) y poder realizar comparaciones con él.

En la parte descriptiva de esta arquitectura se conectarán todos los componentes según la figura 3.22 y se realizará la conversión comentada. Dicho código se implementa según lo siguiente:

```
47
48    BEGIN
49       Conversor: Conv_bcd PORT MAP              --Instancia Conversor BCD
50             (Conversion, BCD_3, BCD_2, BCD_1, BCD_0);
51
52       Contador: Contcuatro PORT MAP             --Instancia Contador Mod4
53             (Clk, Inactivo, Muestreado, Refrescado, Selector);
54
55       Multi: Multcomp PORT MAP                  --Instancia Multiplexor 4bits
56             (Selector, BCD_3, BCD_2, BCD_1, BCD_0, BCD_MUX);
57
58       Decodificador: Decod PORT MAP             --Instancia Decodificador
59             (Selector, AN);
60
61       Conversion <= UNSIGNED (DistFinal);       --Conversion a Numero sin Signo
62
63  END Temporal;
```

Como ya se ha observado en otros circuitos que albergaban componentes o circuitos internos, en este código se aprecia toda la asignación de señales coherentemente con la anterior figura.

Además, como se adelanto, se aprecia el volcado del vector de entrada DistFinal(7) sobre el vector UNSIGNED Conversión(7) mediante una conversión de tipo de objeto.

Para completar el por qué de esta asignación y se pueda entender el funcionamiento global del circuito Multiplexor, antes de comenzar a describir los componentes de este circuito; conviene hacer hincapié en la complementariedad de Multcomp y Decod para llevar a cabo la multiplexión temporal.

Lo más práctico para entender esta asignación es conocer a grandes rasgos el funcionamiento de los componentes Multcomp y Decod. Esto es, conocer sus salidas en función de sus entradas para dejar cerrada la descripción de su diseño en los siguientes epígrafes.

El componente Multcomp tiene un funcionamiento que se basa en que, dependiendo de la entrada que recibe a través del vector Selector(2), se obtiene a la salida un único código BCD de entre los cuatro que tiene a su entrada.

Estos cuatro códigos BCD se llaman BCD_0, BCD_1, BCD_2 y BCD_3. El número que aparece después de cada guión bajo indica la posición del dígito del display en el que debe aparecer dicho código. Esta posición se fija según la numeración de los ánodos de la placa de desarrollo Spartan III, es decir, el dígito de mayor peso será el de mayor número.

El vector Selector(2) puede tomar los valores 0, 1, 2 y 3 en binario. Pues bien, el valor de este vector indicará la cifra a representar en cada momento. De tal forma, si el vector Selector(2) vale 0, el dígito en código BCD que se obtendrá en BCD_MUX a la salida de Multcomp será BCD_0.

Por otra parte, el componente Decod tiene en su salida el vector AN(4) con todos los valores a ‘1’ excepto el correspondiente al dígito a representar que se encuentra a ‘0’. Este vector se conectará a los ánodos de los dígitos de la placa según la numeración de dichos ánodos, es decir, el ánodo del dígito de mayor peso se conectará con el bit de mayor peso del vector AN(4).

De este modo, cuando el vector Selector(2) valga 2, el dígito a representar será el del ánodo de placa nombrado como AN2, esto es, el tercer dígito empezando por la derecha. Este valor 2 del vector Selector(2) hace que el componente Decod obtenga a su salida el vector AN(4) tenga un valor de “1011”, es decir, que el único ‘0’ de este vector sea el tercero empezando por la derecha que es el conectado al ánodo mencionado.

Pero al mismo tiempo y extremadamente sincronizado, el componente Multcomp mostrará en el vector BCD_MUX de su salida el vector de entrada correspondiente al tercer dígito, esto es, el vector BCD_2.

El vector Selector(2) corresponde con el valor de salida de un contador de módulo cuatro que es el componente Contcuatro. Por lo tanto, cuando se llega a 3 se volverá a 0 y así continuamente refrescando todos los dígitos de derecha a izquierda y vuelta a empezar.

Con esto queda detalladamente explicado el proceso de multiplexión interno del bloque Multiplexor para poder comprender a continuación el funcionamiento interno de cada componente de este Multiplexor.

3.4.4.A.1. Contcuatro

En función de lo ya mencionado, este será el componente del bloque Multiplexor que pone en el vector Selector(2) un valor procedente de una cuenta en módulo 4; siendo esta la base de la multiplexión temporal en este bloque.

Sin embargo, también se sabe que este bloque será el encargado de la generación de la señal Refrescado generada a partir de la señal Muestreado y el número de refrescos mínimos estipulados para poder iniciar una nueva medida.

Sin más dilación se muestra la declaración de librerías y arquitectura de este componente llamado *Contcuatro* cuyo código VHDL se encuentra en el archivo *Contcuatro.vhd*:

```
1  LIBRARY IEEE;                    --Librerias
2     USE ieee.std_logic_1164.all;
3     USE ieee.std_logic_arith.all;
4
5
6  ENTITY Contcuatro IS             --Entidad Contador 4 estados
7     GENERIC (DIV_FREQ : POSITIVE   := 5000;         --Division de Frecuencia
8              VISUALIZAR : POSITIVE := 17000000);    --N° Veces de Refresco
9     PORT (Clk, Inactivo, Muestreado : in std_logic; --Entradas y Salidas
10          Refrescado                : out std_logic := '1';
11          Selector                  : out std_logic_vector (1 DOWNTO 0));
12 END Contcuatro;
13
```

La librería necesaria para obtener la funcionalidad descrita en este circuito es la IEEE con los paquetes std_logic_1164 y std_logic_arith para la declaración de tipos de objetos y operaciones.

Ya en la entidad de este componentes se observan dos datos genéricos que son los siguientes:

- *DIV_FREQ:* Este dato obtiene una frecuencia de reloj inferior en el interior del circuito. Se comprobó durante el desarrollo que si la multiplexión se hacía correr a 50 MHz. (oscilador) los caracteres representados se distorsionaban. Por esto dicha multiplexión se realiza a 10 KHz., frecuencia suficiente para aparentar un display uniforme en el ojo humano. Por esto el valor de este dato es 5.000.

- *VISUALIZAR:* Este dato contiene el número de ciclos base, es decir a frecuencia 50 MHz., que se debe refrescar un dato en el display antes de realizar otra medida y perder ese dato para ofrecer cierta estabilidad en la medida y poder apreciar todas ellas. El número 17.000.000 multiplicado por la duración de cada ciclo 1/50.000.000 (50 MHz.) da como resultado 0,34 segundos. Esto quiere decir que

cada medida se mostrará, como mínimo, durante 0,34 segundos en el display.

Las entradas de este componente son las que se comentan a continuación:

- *Inactivo:* Esta señal detiene la multiplexión y resetea todos los recursos de este circuito dejándolo inutilizado mientras que dicha señal se encuentre a nivel alto.

- *Muestreado:* Esta señal interviene en el cálculo del tiempo mínimo de refresco de una medida contando como que la medida se empieza a mostrar a partir de que es guardada en el Registro del Procesador.

Y las salidas de este circuito hacia el exterior y diversos componentes del bloque Multiplexor son las siguientes:

- *Refrescado:* Esta señal se pone a nivel alto cuando un dato ha sido refrescado durante el tiempo mínimo en el display, es decir, durante 0,34 segundos. Además dicha señal se inicializa a nivel alto para que el Secuenciador pueda tomar la primera medida cuando se inicia por primera vez.

- *Selector(2):* Las funciones de este vector se han comentado ya y son de sincronismo para llevar a cabo la multiplexión en el display de la placa.

La arquitectura de este circuito se llama *Modulocuatro* y en ella se implementa un contador en módulo cuatro al que se le han añadido secuencias de control para dividir la frecuencia de funcionamiento y para gestionar el refresco mínimo del display.

No existe parte declarativa en esta arquitectura. Por lo tanto se comienza la misma con su parte descriptiva ocupada en su totalidad por un proceso según se muestra a continuación:

```
14
15  ARCHITECTURE Modulocuatro OF Contcuatro IS   --Modulocuatro
16
17     BEGIN
18        PROCESS (Clk, Inactivo)                  --Proceso de Cuenta
19
20           VARIABLE Cuenta : UNSIGNED (1 DOWNTO 0);  --Variables de Cuenta
21           VARIABLE d : INTEGER RANGE 0 TO DIV_FREQ;
22           VARIABLE Aux : INTEGER RANGE 0 TO VISUALIZAR;
```

Como se observa en el código, este proceso tiene en su lista de sensibilidad la señal de reloj Clk y la señal de neutralización asíncrona llamada Inactivo (--Proceso de Cuenta). En la parte declarativa del proceso se declaran

tres variables que ayudarán a la implementación del contador así como la de los elementos adicionales añadidos a este (--Variables de Cuenta).

La variable Cuenta(2) de tipo UNSIGNED es la que variará de 0 a 3 llevando la cuenta y el control de la multiplexión. La variable d es de tipo INTEGER desde 0 hasta el dato genérico DIV_FREQ y, como se intuye, generará la división de la frecuencia del circuito. La variable Aux de tipo INTEGER desde 0 hasta VISUALIZAR generará el tiempo mínimo de refresco.

El proceso se sigue desarrollando acorde con el siguiente código:

```
23
24          BEGIN
25             IF Inactivo = '1' THEN                 --Estado de Inactividad
26                Refrescado <= '1';
27                d := 0;
28                Aux := 0;
29                Cuenta := (OTHERS => '0');
30
31             ELSIF Clk 'EVENT AND Clk = '1' THEN     --Flanco Ascendente Clk
32                IF Muestreado = '1' THEN                --Control de Refresco
33                   Aux := 0;
34                   Refrescado <= '0';
35                ELSIF Aux < VISUALIZAR THEN
36                   Aux := Aux + 1;
37                ELSE
38                   Refrescado <= '1';
39                END IF;
40
41                IF d = DIV_FREQ THEN                  --Control de Multiplexion
42                   d := 0;                             --Cuentas Prim. y Sec.
43                   IF Cuenta = 3 THEN
44                      Cuenta := (OTHERS => '0');
45                   ELSE
46                      Cuenta := Cuenta + 1;
47                   END IF;
48                ELSE
49                   d := d + 1;
50                END IF;
51
52             END IF;
53
54             Selector <= STD_LOGIC_VECTOR (Cuenta);  --Conversion a Vector Binario
55
56          END PROCESS;                            --Fin de Proceso de Cuenta
57
58    END Modulocuatro;
```

La primera sentencia que se aprecia en este proceso es de tipo IF y provoca que si la señal Inactivo se encuentra a nivel alto la señal de salida Refrescado se ponga a nivel alto para que el Secuenciador pueda iniciar futuras mediciones, así como que se reseteen el resto de variables d, Aux y Cuenta(2) para evitar la multiplexión y forzar la inactividad (--Estado de Inactividad).

Si la variable Inactivo se encuentra a nivel bajo y se produce un flanco ascendente de reloj en la señal Clk, se ejecutará el código donde se gestiona toda la cuenta y multiplexión (--Flanco Ascendente Clk).

Este código se divide en dos bloques, el de control de refresco y el de control de multiplexión; identificados por los comentarios del mismo nombre.

En el control de refresco se utiliza la variable Aux. Cuando la señal Muestreado se encuentra a nivel alto al guardar un dato en el Registro del Procesador dicha variable Aux se resetea y la señal Refrescado se pone a nivel bajo para comenzar una cuenta de tiempo mínimo de refresco.

Cuando la señal Muestreado ya se encuentre a nivel bajo siempre que la variable Aux sea menor al dato genérico VISUALIZAR dicha variable se incrementará en uno.

Y finalmente, cuando se alcance el dato VISUALIZAR, al pasar 17.000.000 ciclos desde que se comenzó a multiplexar la última medida, la señal Refrescado se pondrá a nivel alto. De tal forma quedará gestionado todo el proceso de control del refresco.

En cuanto al control de multiplexión, se realizará con la variable d. Cuando d alcance el valor máximo de división de frecuencia, es decir, DIV_FREQ; dicha variable d será reiniciada para comenzar otro proceso de división de frecuencia. Además, en ese momento se incrementará la cuenta en la variable Cuenta(2); teniendo en cuenta que si esta variable vale 3 el siguiente número será 0.

Por otra parte, si d todavía no alcanza al número DIV_FREQ, a esta variable d se le incrementará su valor en uno para gestionar el proceso de división de frecuencia.

Una vez se cierre el bloque de la sentencia IF-ELSIF, se asignará a la señal de salida Selector(2) la variable de trabajo para la cuenta llamada Cuenta(2).

Por último se cierra el proceso y se finaliza la descripción de la arquitectura Modulocuatro según la funcionalidad especificada anteriormente.

Esta descripción meramente secuencial por proceso, explica el funcionamiento de la multiplexión del display y del refresco mínimo de cara al secuenciador y teniendo en cuenta la especificación del sistema de medida de tres veces por segundo.

3.4.4.A.2. Decod

Este componente, cuya función en el componente Multiplexor ya se ha descrito, posee el comportamiento básico del componente electrónico estándar llamado decodificador. De ahí su nombre *Decod*.

Concretamente, el componente sería un decodificador de código de 2 a 4. Con lo anteriormente explicado y el funcionamiento de dicho componente electrónico, según la entrada de este circuito se activará un solo bit (a nivel bajo) del vector de salida de 4 bits, dependiendo del vector de entrada de 2 bits.

Con esta funcionalidad bastará para poder gestionar la activación de cada dígito en orden y en base a la posición del código de 7 segmentos que se esté sacando a los cátodos del display.

Según lo comentado, el código VHDL se encuentra en el archivo *Decod.vhd* y se implementa según lo siguiente:

```
1  LIBRARY IEEE;                    --Librerias
2     USE ieee.std_logic_1164.all;
3
4
5  ENTITY Decod IS                  --Entidad Decod: Decodificador
6     PORT (Selector : in std_logic_vector (1 DOWNTO 0); --Entradas y Salidas
7           AN       : out std_logic_vector (3 DOWNTO 0));
8  END Decod;
9
```

Este fragmento de código muestra la declaración de librerías y paquetes del componente Decod, así como la descripción de su entidad.

En cuanto al paquete declarado es el std_logic_1164 para la declaración de tipos de objetos std_logic y sus derivados. Este paquete pertenece a la librería IEEE.

La entrada de este circuito es el vector Selector(2), fruto de la cuenta del componente Contcuatro cuyo funcionamiento ya se conoce. La señal de salida AN(4) se conectará a cada uno de los ánodos de cada dígito del display para generar la multiplexión temporal.

Lo siguiente en la descripción de este componente es su arquitectura, la cual se llama *Anodos* y se muestra a continuación:

```
10
11  ARCHITECTURE Anodos OF Decod IS     --Anodos
12
13     BEGIN
14        PROCESS(Selector)               --Proceso de Decodificacion
15
16           BEGIN
17              CASE Selector IS
18                 WHEN "00" => AN <= "1110";
19                 WHEN "01" => AN <= "1101";
20                 WHEN "10" => AN <= "1011";
21                 WHEN "11" => AN <= "0111";
22                 WHEN OTHERS => AN <= "0111";
23              END CASE;
24
25           END PROCESS;                --Fin de Proc. de Decod.
26
27  END Anodos;
```

Como se aprecia a primera vista, esta arquitectura no posee parte declarativa y su parte descriptiva consiste en un proceso cuya lista de sensibilidad esta formada por la entrada al circuito Selector(2) (--Proceso de Decodificación).

En la parte descriptiva de dicho proceso se encuentra el código VHDL habitual para la síntesis de un decodificador. Esto es una sentencia CASE en la que se fija una salida AN(4) para cada entrada Selector(2).

Para que el componente Decod funcione según lo establecido, cuando la entrada sea “00”, el bit que se encuentre a ‘0’ será el de menor peso. El bit de mayor peso del vector AN(4) se encontrará a ‘0’ cuando el vector Selector(2) sea “11”.

En dicha sentencia CASE se introduce un código WHEN OTHERS por seguridad aunque se sabe que el vector Selector(2) solo puede tomar valores de 0 a 3.

Con esto queda definido el componente Decod, el cual es un componente electrónico estándar y cuya implementación se antoja sencilla como se ha podido comprobar.

3.4.4.A.3. Multcomp

Al igual que el anterior componente comentado, este componente tiene el funcionamiento de un multiplexor típico. De tal forma, dependiendo de las entradas de selección, comunica una de sus entradas de datos con la única salida que posee.

En concreto, el multiplexor que en este proyecto se implementa tiene cuatro entradas de datos de vectores de 4 bits cada uno. Dadas sus cuatro entradas, necesitará un vector de entrada de selección de 2 bits para poder seleccionar todas las entradas.

Analizando el código VHDL de este circuito, contenido en *Multcomp.vhd*, se observa la siguiente descripción de librerías y entidad:

```
1  LIBRARY IEEE;                    --Librerias
2     USE ieee.std_logic_1164.all;
3
4
5  ENTITY Multcomp IS               --Entidad Multcomp: Multiplexor 4 a 1
6     PORT (Selector                   : in std_logic_vector (1 DOWNTO 0); --Entradas y Salidas
7           BCD_3, BCD_2, BCD_1, BCD_0 : in std_logic_vector (3 DOWNTO 0);
8           BCD_MUX                    : out std_logic_vector (3 DOWNTO 0));
9  END Multcomp;
10
```

La librería y paquete declarados en este circuito son los mismos que en el caso del decodificador. Su función es sencilla de implementar y solamente se necesita el paquete std_logic_1164 de la librería IEEE.

Como se ha descrito anteriormente, las entradas de este componente serán BCD_0, BCD_1, BCD_2 y BCD_3; cada una de 4 bits. La salida de dicho circuito se llama BCD_MUX con el mismo número de bits. El canal de selección se llama Selector, con una longitud de 2 bits.

Para que este componente tenga la funcionalidad descrita, la arquitectura que se desarrolla es la siguiente, la cual se llama *Selección*:

```
11
12 ARCHITECTURE Seleccion OF Multcomp IS  --Seleccion
13
14    BEGIN
15       PROCESS(Selector, BCD_3, BCD_2, BCD_1, BCD_0)   --Proceso de Multiplexion
16
17          BEGIN
18             CASE Selector IS
19                WHEN "00" => BCD_MUX <= BCD_0;
20                WHEN "01" => BCD_MUX <= BCD_1;
21                WHEN "10" => BCD_MUX <= BCD_2;
22                WHEN "11" => BCD_MUX <= BCD_3;
23                WHEN OTHERS => BCD_MUX <= "1111";
24             END CASE;
25
26          END PROCESS;                              --Fin de Proc. de Mult.
27
28 END Seleccion;
```

Como se observa en el código, esta arquitectura carece de parte declarativa y en su parte descriptiva solamente se implementa un proceso cuya lista de selección consta de todas las entradas del circuito. Aquí es donde se dará la multiplexión (--Proceso de Multiplexión).

En la parte descriptiva del proceso de multiplexión se aprecia una sentencia de tipo CASE que conmuta las acciones en función de la entrada Selector(2). De tal modo, la decisión de qué vector BCD_X(4) se asigna a la salida BCD_MUX(4) depende del número que contenga Selector(2). Si dicho vector contiene el número 2 en binario, el vector que se comunica con la salida es BCD_2 y así consecutivamente.

Este proceso y su sentencia CASE es suficiente para codificar el circuito Multcomp, que realice todas las funciones necesarias para completar la multiplexión tal y como se ha explicado anteriormente.

3.4.4.A.4. Conv_bcd

Este componente del conjunto Multiplexor es el más complejo de entre todos los que ya se han descrito. Su función es la conversión de un número en binario que se recibe del Procesador a cuatro números en código BCD *(Binary Coded Decimal)*.

Realmente, el Procesador únicamente aporta un número que va de 0 a 100 sin ser el 100 representable (Es el código de error). Por lo tanto, con la conversión a dos números BCD sería suficiente para descomponer este número.

Sin embargo, para todo el circuito Conversor se pretende hacer una plataforma lo más estándar que sea posible con vistas a reutilizarla realizando pocas variaciones.

De tal forma, para cualquier otra representación en el display que no sea la del Medidor, es posible que se necesite la conversión de cuatro dígitos BCD así como la multiplexión de los mismos. Este código se podrá adaptar fácilmente a ese sistema.

Es por esto que, por el momento, los códigos BCD asignados con la conversión del número que procede del Procesador son BCD_3(4) y BCD_2(4). Los otros dos números BCD_1(4) y BCD_0(4) tendrán siempre un valor de “0000”, obteniendo el valor “1111” en caso de error.

Sin más apuntes, se pasa a describir el código VHDL que implementa la funcionalidad comentada y que está contenido en el archivo *Conv_bcd.vhd*. Se comienza con la entidad del circuito:

```
1  LIBRARY IEEE;                        --Librerias
2     USE ieee.std_logic_1164.all;
3     USE ieee.std_logic_arith.all;
4     USE ieee.std_logic_unsigned.all;
5
6
7  ENTITY Conv_bcd IS                   --Entidad Conversor Binario-BCD
8     GENERIC (NUMBITS  : POSITIVE := 7);            --Numero de Bits
9     PORT (DistFinal : in UNSIGNED ((NUMBITS-1) DOWNTO 0); --Entradas y Salidas
10          BCD_3, BCD_2, BCD_1, BCD_0 : out std_logic_vector (3 DOWNTO 0));
11 END Conv_bcd;
12
```

De nuevo, en este fragmento de código se observa que se vuelven a usar los tres paquetes descritos de la librería IEEE. Estos paquetes son std_logic_1164, std_logic_arith y std_logic_unsigned. Dichos paquetes sirven para definiciones de tipos de datos, conversiones de tipos de datos y operaciones.

En cuanto a la entidad, se observa un único dato genérico llamado NUMBITS que se utiliza para definir la longitud de la entrada DistFinal(7), por lo tanto su valor es de siete.

La única entrada de este circuito es la que ya comentada DistFinal(7) y que proviene del valor numérico sin precisión generado en el Procesador.

Las salidas del bloque Conv_bcd son también familiares ya que son los cuatro códigos BCD_0(4), BCD_1(4), BCD_2(4) y BCD_3(4) que se irán a multiplexar al componente Multcomp. De tal forma, todas estas salidas tendrán una longitud de 4 bits.

La arquitectura de este circuito conlleva alguna dificultad más que su entidad. El nombre de dicha arquitectura es *Codificador_bcd* y su código se muestra seguidamente:

```
13
14 ARCHITECTURE Codificador_bcd OF Conv_bcd IS  --Codificador_bcd
15
16    SIGNAL BCD_D, BCD_U : STD_LOGIC_VECTOR (3 DOWNTO 0);  --Señales de Conv.
17
18    BEGIN
19    PROCESS (DistFinal)                     --Proceso de Conversion
20
21       VARIABLE Valor : INTEGER RANGE 0 TO 100;  --Declaracion Variable Trabajo
22
23       BEGIN
24          Valor := CONV_INTEGER (DistFinal(6 DOWNTO 0));  --Inicializacion Variable
25          BCD_1 <= "0000";                     --Display 1 y 0 a Cero
26          BCD_0 <= "0000";
```

Lo primero que se aprecia en este fragmento de código es la parte declarativa de la arquitectura. En esta parte se declaran dos señales de trabajo llamadas BCD_D(4) y BCD_U(4), ambas de 4 bits de longitud y de tipo std_logic_vector. En estas señales se guardan los valores de BCD_3(4) y BCD_(2) durante el proceso de conversión, es decir, las Decenas y Unidades de la cifra que se recibe del Procesador.

Seguidamente comienza la parte descriptiva de la arquitectura con un proceso cuya lista de sensibilidad la ocupa la entrada del circuito DistFinal(7) (--Proceso de Conversión).

En la parte declarativa del proceso aparece una variable de trabajo para asignarle posteriormente la entrada DistFinal(7). Esta variable de trabajo se llama Valor, es de tipo INTEGER y tiene un rango de 0 a 100, justo el mismo que posee dicha entrada.

Una vez en la parte descriptiva del proceso se inicializa la variable Valor con el dato DistFinal(7) como ya se anticipó. Este se realiza mediante una conversión de tipos de objetos a INTEGER. También se inicializan a “0000” las salidas BCD_1(4) y BCD_0(4).

La arquitectura sigue desarrollándose con el siguiente código VHDL dentro de la parte descriptiva del proceso:

```
27
28          CASE Valor IS                              --Caso Decenas
29             WHEN 0 TO 9 => BCD_D <= "0000";
30             WHEN 10 TO 19 => BCD_D <= "0001";
31                              Valor := Valor - 10;
32             WHEN 20 TO 29 => BCD_D <= "0010";
33                              Valor := Valor - 20;
34             WHEN 30 TO 39 => BCD_D <= "0011";
35                              Valor := Valor - 30;
36             WHEN 40 TO 49 => BCD_D <= "0100";
37                              Valor := Valor - 40;
38             WHEN 50 TO 59 => BCD_D <= "0101";
39                              Valor := Valor - 50;
40             WHEN 60 TO 69 => BCD_D <= "0110";
41                              Valor := Valor - 60;
42             WHEN 70 TO 79 => BCD_D <= "0111";
43                              Valor := Valor - 70;
44             WHEN 80 TO 89 => BCD_D <= "1000";
45                              Valor := Valor - 80;
46             WHEN 90 TO 99 => BCD_D <= "1001";
47                              Valor := Valor - 90;
48             WHEN OTHERS => BCD_D <= "1111";            --Gestion de Errores
49          END CASE;
50
51          CASE Valor IS                              --Caso Unidades
52             WHEN 0 => BCD_U <= "0000";
53             WHEN 1 => BCD_U <= "0001";
54             WHEN 2 => BCD_U <= "0010";
55             WHEN 3 => BCD_U <= "0011";
56             WHEN 4 => BCD_U <= "0100";
57             WHEN 5 => BCD_U <= "0101";
58             WHEN 6 => BCD_U <= "0110";
59             WHEN 7 => BCD_U <= "0111";
60             WHEN 8 => BCD_U <= "1000";
61             WHEN 9 => BCD_U <= "1001";
62             WHEN OTHERS => BCD_U <= "1111";
63                            BCD_1 <= "1111";
64                            BCD_0 <= "1111";         --Gestion de Errores
65          END CASE;
```

En este fragmento de código se observan dos sentencias CASE complementarias y muy semejantes a las comentadas anteriormente en el bloque Redondeador del circuito Procesador.

En el conjunto de la primera sentencia se fija el valor de las decenas en la señal de trabajo BCD_D(4). Para ello, se introducen rangos de comparación de 10 en 10 números, localizando las decenas de la cifra convertir (--Caso Decenas).

Cuando se localiza el intervalo de decena en la que se encuentra la variable Valor, se le asigna la decena correspondiente a la señal BCD_D(4). Además, también se le resta a la variable Valor el número de decenas inmediatamente inferior al rango en el que se encuentra y se guarda en la misma variable, quedando únicamente las unidades.

En la segunda sentencia se comparan las unidades de la variable Valor sin rangos asignando directamente a BCD_U(4) el número correspondiente a la unidad que coincida (--Caso Unidades).

En cuanto a la gestión del mensaje de error, en la primera sentencia CASE, el código de error 100 hace que se ejecute la línea WHEN OTHERS poniendo BCD_D(4) a “1111” (--Gestión de Errores). Esta cifra no existe en código BCD pero se utilizará para mostrar el mensaje de error en el display.

En la segunda sentencia CASE, el hecho de que Valor no se encuentre entre 0 y 9 provoca que BCD_U(4) tome también el valor “1111”, y además se modifican las señales BCD_1(4) y BCD_0(4) con el mismo valor (--Gestión de Errores).

Para finalizar el proceso se deben asignar las señales de trabajo BCD_D(4) y BCD_U(4) a las señales de salida BCD_3(4) y BCD_2(4), respectivamente. Esto se desarrolla en el siguiente código:

```
66
67          BCD_3 <= BCD_D;                         --Asignacion a Digitos de Display
68          BCD_2 <= BCD_U;
69
70        END PROCESS;                            --Fin de Proceso de Conversion
71
72  END Codificador_bcd;
```

Finalmente se cierra el proceso, que no se volverá a ejecutar hasta un cambio de valor de DistFinal(7); así como la arquitectura de este circuito, que cumple con todas las especificaciones comentadas anteriormente.

3.4.4.B. Conv_Seg

Este es el otro gran bloque del componente Conversor. También es el último componente que existe entre el display de la placa de desarrollo Spartan III y el diseño VHDL de la FPGA. Por lo tanto, este circuito debe dar ya las señales concretas al display para que este las represente.

La función del componente *Conv_Seg* implica la conversión de cualquier número en código BCD al código de 7 segmentos con punto decimal del display de la placa.

Además, este módulo cuenta con la información de la precisión de la medida así como la posición del display que se representa en cada momento. Esto le permitirá generar símbolos de precisiones y mostrar del mensaje de error.

Para comprender el desarrollo de este componente, hay que recordar en todo momento las especificaciones de la placa Spartan III en cuanto a lo que el display se refiere. De tal modo, es importante recordar que los segmentos del display se activan a nivel bajo, por ejemplo.

El código VHDL que sintetiza este circuito se encuentra en el archivo *Conv_Seg.vhd* y se comenta a continuación:

```
1  LIBRARY IEEE;                    --Librerias
2     USE ieee.std_logic_1164.all;
3
4
5  ENTITY Conv_seg IS                --Entidad Conv_Seg
6     PORT (BCD_MUX           : in std_logic_vector(3 DOWNTO 0);--Entradas y Salidas
7           AN                : in std_logic_vector(3 DOWNTO 0);
8           mt_ncmt, Inactivo : in std_logic;
9           SieteSeg          : out std_logic_vector(7 DOWNTO 0));
10 END Conv_seg;
11
```

El único paquete declarado para hacer funcionar este circuito es el std_logic_1164 para la declaración de tipos std_logic. La librería a la que pertenece este paquete es la IEEE.

En cuanto a la entidad de este componente se aprecia la carencia de datos genéricos y se empieza directamente la descripción de entradas y salidas del mismo, coherentes con la figura 3.21. En cuanto a las entradas de este circuito se encuentran las siguientes:

- *BCD_MUX(4):* Este vector proviene del bloque Multiplexor y, como se ha desarrollado, contiene un dato en código BCD multiplexado temporalmente mediante la señal AN (4).

- *AN(4):* Este conjunto de señales agrupadas indica que dígito del display está activo a la par que el número BCD que se está multiplexando.

- *mt_ncmt:* Esta señal proviene del Procesador siendo el bit de mayor peso del vector de salida de este bloque. Dicho bit codifica la precisión en la que se tiene que representar el dato.

- *Inactivo:* La función de esta señal ha sido ya comentada y para este circuito se neutralizará toda salida al display.

La única salida de este bloque es el vector SieteSeg(8) el cual contiene el dato multiplexado en cada momento en código 7 segmentos con punto decimal para ser directamente representado en el dígito del display que indique la señal AN(4). Este vector puede contener el equivalente a números, símbolos de precisión, o bien el letras del mensaje de error.

La arquitectura correspondiente a este circuito es la llamada *Tabla* que se muestra a continuación:

```
12
13  ARCHITECTURE Tabla OF Conv_seg IS   --Tabla
14
15     SIGNAL Numero : std_logic_vector (6 DOWNTO 0);  --Señales Internas
16     SIGNAL Punto  : std_logic;
17
18     BEGIN
19        PROCESS (BCD_MUX, AN, mt_ncmt, Inactivo)  --Proceso de Conversion de Numeros + Error
20
21           BEGIN
22              IF Inactivo = '1' THEN                          --Estado de Inactividad
23                 Numero (6 DOWNTO 0) <= "1111111";
```

En la parte declarativa de esta arquitectura aparecen dos señales de trabajo para ayudar a generar el vector de salida SieteSeg(8); estas son Numero(7) y Punto en las que se guardará el carácter a representar y el punto decimal, respectivamente.

Seguidamente comienza la parte descriptiva de la arquitectura con el proceso de conversión de números o símbolos y error (--Proceso de Conversión de Números + Error). Para generar el resultado comentado es necesario que este proceso tenga en su lista de sensibilidad todas las entradas del circuito. Cuando cualquiera de ellas cambie cambiarán las condiciones de representación.

En cuanto se entra en la parte descriptiva del proceso, el cual no posee parte declarativa, se encuentra una secuencia IF. Esta secuencia implica que si la señal Inactivo se encuentra a nivel alto la señal Numero(7) tomará el valor “1111111”. Como los segmentos del display de la placa funcionan a nivel bajo,

esta asignación apaga los 7 segmentos de todos los dígitos del display, que es lo que se desea (--Proceso de Inactividad).

A continuación se prosigue con este código:

```
24
25            ELSE
26               CASE BCD_MUX IS
27                     -- DCBA
28                  WHEN "0000" => IF mt_ncmt = '1' THEN          --Si son metros se muestra M
29                                    IF AN (1) = '0' THEN
30                                       Numero <= "0011001";
31                                    ELSIF AN (0) = '0' THEN
32                                       Numero <= "0001101";
33                                    ELSE
34                                       Numero <= "0000001";
35                                    END IF;
36
37                                 ELSIF mt_ncmt = '0' THEN       --Si son centimetros se muestra cm
38                                    IF AN (1) = '0' THEN
39                                       Numero <= "1110010";
40                                    ELSIF AN (0) = '0' THEN
41                                       Numero <= "1101010";
42                                    ELSE
43                                       Numero <= "0000001";
44                                    END IF;
45
46                                 ELSE
47                                    Numero <= "0000001";
48
49                                 END IF;
```

La sentencia ELSE significa que, si la señal Inactivo se encuentra a nivel bajo, se ejecuta la parte del código que la misma contiene. Dicho código consta de una gran sentencia CASE donde se chequean los valores que contiene el vector de entrada BCD_MUX(4).

En este fragmento de código se encuentra el caso de que dicho vector de entrada sea “0000”. Recordando los componentes anteriormente descritos esto puede ser por varios motivos.

Se pueden estar multiplexando los códigos BCD_1(4) y BCD_0(4) al no haberse producido ningún error, en cuyo caso se tendrán que dibujar los símbolos correspondientes a la precisión dada en esas posiciones del display.

Pero también puede ser el caso que se desee mostrar el número 0 en las posiciones 2 o 3 del display. Ambos casos son tenidos en cuenta.

La primera sentencia IF gestiona el caso de la representación del símbolo de metros a través de la condición de que la señal mt_ncmt se encuentre a nivel alto (--Si son metros se muestra M). Dentro de esta condición, si se está representando el dígito 1 (AN[1] = ‘0’) o el dígito 0 (AN[0] = ‘0’); se tiene que mostrar en el display el símbolo de la figura 3.23. Si no se están representando esos dígitos se mostrará un 0, como es de esperar.

A continuación se muestra la figura 3.23:

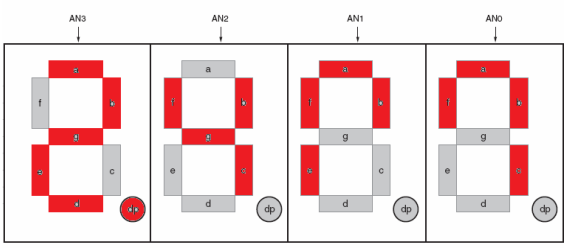


Figura 3.23: Display con medida en precisión de metros

Si la primera sentencia IF no se verifica y además la condición de que la señal mt_ncmt se encuentre a nivel bajo, se ejecuta la sentencia ELSIF donde se gestiona el caso de la representación del símbolo de centímetros (--Si son centímetros se muestra cm). Si se resetea el dígito 1 (AN[1] = ‘0’) o el dígito 0 (AN[0] = ‘0’); se tiene que mostrar en el display el siguiente símbolo:

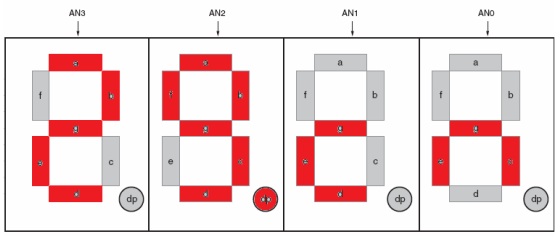


Figura 3.24: Display con medida en precisión de centímetros

Si no se da ninguno de los dos casos anteriores, para cerrar el bloque IF, se introduce como precaución la sentencia ELSE donde se mostraría un 0 de ejecutarse.

A continuación se prosigue con la asignación de código 7 segmentos a la señal Numero(7), ahora ya asignando directamente los valores según el código BCD recibido:

```
50
51                      -- DCBA                abcdefg
52                  WHEN "0001" => Numero <= "1001111";
53                  WHEN "0010" => Numero <= "0010010";
54                  WHEN "0011" => Numero <= "0000110";
55                  WHEN "0100" => Numero <= "1001100";
56                  WHEN "0101" => Numero <= "0100100";
57                  WHEN "0110" => Numero <= "0100000";
58                  WHEN "0111" => Numero <= "0001111";
59                  WHEN "1000" => Numero <= "0000000";
60                  WHEN "1001" => Numero <= "0000100";
61                  WHEN OTHERS => CASE AN IS  --Si BCD > 9, error
62                                     WHEN "1110" => Numero <= "1100010";
63                                     WHEN "1101" => Numero <= "1111010";
64                                     WHEN "1011" => Numero <= "1111010";
65                                     WHEN "0111" => Numero <= "0110000";
66                                     WHEN OTHERS => Numero <= "1111111";
67                                 END CASE;
68              END CASE;
69
70          END IF;
71
72      END PROCESS;                                --Fin de Proc. Conv. N + E
```

Lo más destacado de este código es la asignación del mensaje de error cuando la señal BCD_MUX(4) de entrada se sale del rango de este código (de 0 a 9). Esto se realiza mediante una sentencia CASE anidada en la sentencia superior. De tal forma, dependiendo del vector AN(4) se representará una letra u otra hasta completar el mensaje “Erro” (--Si BCD >9, error).

Con esto finaliza la sentencia CASE de asignación de Numero (7) a través del valor del vector BCD_MUX(4). También finaliza el bloque de la sentencia IF así como el proceso de gestión de números y símbolos.

Todo este proceso genera en el display de la placa Spartan III la variedad de caracteres mostrados en la figura 3.25:

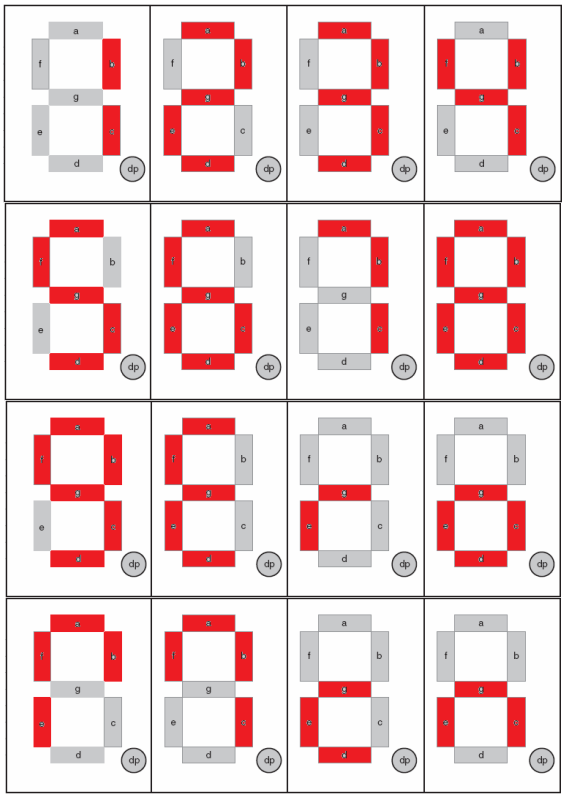


Figura 3.25: Caracteres implementados en el circuito Conv_Seg

El orden en el que se muestran los caracteres den display en la figura 3.25 es el siguiente: 1, 2, 3, 4, 5, 6, 7, 8, 9, E, r, o, M, cm.

La arquitectura de este componente prosigue con otro proceso con la misma lista de sensibilidad y que por lo tanto se ejecutará a la vez que el anterior proceso descrito pero paralelamente con este, así correrán a mayor velocidad.

Este es el proceso de gestión del punto decimal y su código se muestra a continuación:

```
73
74      PROCESS (BCD_MUX, AN, mt_ncmt, Inactivo)  --Proceso de Gestion de Punto
75
76         BEGIN
77            IF Inactivo = '1' THEN                          --Estado de Inactividad
78               Punto <= '1';
79
80            ELSE
81               IF BCD_MUX <= "1001" THEN                    --Si no hay error BCD_MUX <= 9
82                  IF mt_ncmt = '1' THEN                        --Si son mt. D.D MM
83                     IF AN (3) = '0' THEN
84                        Punto <= '0';
85                     ELSE
86                        Punto <= '1';
87                     END IF;
88
89                  ELSIF mt_ncmt = '0' THEN                     --Si son cmt. DD. cm
90                     IF AN (2) = '0' THEN
91                        Punto <= '0';
92                     ELSE
93                        Punto <= '1';
94                     END IF;
95                  END IF;
96
97               ELSE
98                  Punto <= '1';
99
100              END IF;
101
102           END IF;
103
104        END PROCESS;                                  --Fin de Proc. de Gestion de Punto
105
106     SieteSeg (7 DOWNTO 1) <= Numero (6 DOWNTO 0);   --Asignacion Señales de Salida
107     SieteSeg (0) <= Punto;
108
109  END Tabla;
```

Nada más comenzar la parte descriptiva de este proceso, se ejecuta una sentencia IF semejante a la del proceso anterior siempre y cuando la señal Inactivo se encuentre a nivel alto. Si se da esta condición la señal interna Punto se pone a nivel alto apagando dicho marcador (--Estado de Inactividad).

Si la señal Inactivo no se encuentra a nivel alto se ejecuta la sentencia ELSE apareada a la anterior. En ese caso, se chequea con otra secuencia IF si el vector de entrada BCD_MUX(4) se encuentra en los límites del código BCD. Si es este el caso, el código BCD no será de error y será, por lo tanto, una medida a representar (--Si no hay error BCD_MUX <= 9).

De tal modo, se observa como tras la anterior condición, se establece ahora la separación del caso metros y centímetros mediante el nivel alto o bajo de la señal mt_ncmt, respectivamente.

Según se aprecia en la figura 3.23, cuando se tiene una medida en precisión de metros, al representar dos cifras; se obtendrá la primera cifra indicadora de metros seguida de un punto decimal y un decimal correspondiente a los decímetros.

Por esto, si se verifica que la señal mt_ncmt se encuentra a nivel alto indicando precisión de metros, se pondrá la señal Punto a ‘0’ cuando se esté multiplexando el tercer dígito del display. Esto quiere decir que se encenderá el punto decimal que separa metros de decímetros (--Si son mt. D.D MM).

En cambio y según la figura 3.24, cuando la medida se da en centímetros, el punto decimal aparecerá después de la cifra de unidades de centímetros; separando el valor numérico del símbolo indicador de precisión.

Esta representación conlleva a que cuando la señal mt_ncmt se encuentra a nivel bajo la señal Punto vale ‘0’; solamente si la cifra que se está multiplexando corresponde al segundo dígito del display (--Si son cmt. DD. cm).

Todo esto se gestiona con las sentencias IF-ELSE que se observan de la línea 82 a la 95 y que solo se ejecutan cuando el código BCD que se este multiplexando sea un número válido según este código.

En el caso de que el número multiplexado sea mayor a 9 se obtendrá en el display un mensaje de error. En este caso no se debe representar ningún punto en dicho display y por este motivo la señal Punto valdrá siempre ‘1’.

Con esto se puede cerrar la sentencia IF y el proceso de gestión del punto decimal en el que se encuentra. Con estos dos procesos este circuito ya posee la funcionalidad deseada, sin embargo, queda la asignación de las señales de trabajo a la señal de salida.

Esta asignación se realiza en dos pasos sobre el vector de salida SieteSeg(8). En el primer paso se asigna el vector Numero(7) a las siete posiciones de mayor peso del vector SieteSeg(8). Seguidamente se introduce el bit Punto en el bit de menor peso del vector de salida (--Asignación de Señales de Salida).

De tal forma se obtiene el vector de salida compacto SieteSeg(8) que contiene cada uno de los códigos en 7 segmentos multiplexados para generar caracteres o cifras. Con este módulo, finalmente se cierra la utilización del display de 4 dígitos de 7 segmentos de la placa Spartan III para este proyecto.

3.4.5. Monoestable

Este componente, encuadrado como bloque en el Medidor, tiene la función de monoestable tal y como su nombre indica. Esto significa que, recibiendo una señal que pasa de '0' a '1' lógicos durante un tiempo indefinido, genera una salida que pasa a la vez de '0' a '1' pero que se mantiene a nivel alto un tiempo determinado.

La generación de esta señal es imprescindible para que, solamente se emitan ondas de ultrasonido durante un periodo corto y determinado, aunque la señal que excita a este componente permanezca durante toda la medida a nivel alto.

El código VHDL que da lugar a este componente se encuentra en el archivo *Monoestable.vhd*. Dicho código se comenta a continuación:

```
1  LIBRARY IEEE;                    --Librerias
2     USE ieee.std_logic_1164.all;
3     USE ieee.std_logic_arith.all;
4     USE ieee.std_logic_unsigned.all;
5
6
7  ENTITY Monoestable IS            --Entidad Monoestable
8     GENERIC (NUM_CICLOS : POSITIVE := 200000);   --Numero de Ciclos
9     PORT (Ping, Clk     : in std_logic;          --Entradas y Salidas
10          Pulso         : out std_logic);
11 END Monoestable;
12
```

Como se aprecia en este fragmento de código, para este componente se declaran los tres paquetes utilizados en este proyecto de la librería IEEE. Estos paquetes son el std_logic_1164, el std_logic_arith y el std_logic_unsigned.

Seguidamente aparece la entidad del circuito. Dicha entidad contiene un dato genérico llamado NUM_CICLOS, que fija el número de ciclos de reloj que permanecerá la salida del circuito a nivel alto antes de volver a su estado de reposo.

En cuanto a las salidas del circuito se encuentran la señal de disparo llamada Ping, que actuará cuando se ponga a nivel alto, y la señal de reloj Clk.

La única salida de este circuito es la llamada Pulso, cuya funcionalidad ya se ha descrito anteriormente y depende del dato genérico NUM_CICLOS.

El código VHDL de este circuito prosigue con la arquitectura de la entidad comentada. Esta arquitectura se llama *Generación* y se implementa según lo siguiente:

```
13
14  ARCHITECTURE Generacion OF Monoestable IS --Tren de Pulsos
15
16     BEGIN
17        PROCESS (Clk, Ping)                 --Proceso de Generacion
18
19           VARIABLE Cuenta : INTEGER RANGE 0 TO NUM_CICLOS;   --Variables de Cuenta
20           VARIABLE Disparo, Fin_Cuenta, Est_Anterior : std_logic;
21
22           BEGIN
23              IF Clk 'EVENT AND Clk = '0' THEN    --Flanco Decencente Reloj
24
25                 IF Est_Anterior = '0' AND Ping = '1' THEN --Flanco Ascendente Pulsador
26                    Disparo := '1';
27                 ELSIF Fin_Cuenta = '1' THEN               --Fin Pulso
28                    Disparo := '0';
29                 END IF;
30
31                 IF Disparo = '1' THEN                     --Actuacion Flanco Asc. Puls.
32
33                    IF Cuenta < NUM_CICLOS THEN            --Emision Pulso durante NUM_CICLOS
34                       Cuenta := Cuenta + 1;
35                       Fin_Cuenta := '0';
36                       Pulso <= '1';
37                    ELSE                                   --Finalizar Pulso y Prepara sgte.
38                       Cuenta := 0;
39                       Fin_Cuenta := '1';
40                       Pulso <= '0';
41                    END IF;
42
43                 END IF;
44
45                 Est_Anterior := Ping;
46
47              END IF;
48
49           END Process;                  --Fin Proceso Generacion
50
51  END Generacion;
```

Esta arquitectura no posee parte declarativa y en su parte descriptiva cuenta únicamente con un proceso. En la lista de sensibilidad de este proceso se encuentran las dos entradas de este bloque Clk y Ping.

En la parte declarativa de este proceso se encuentra una variable llamada Cuenta que es de tipo INTEGER desde 0 al dato genérico NUM_CICLOS. Esta variable sirve para llevar la cuenta de número de ciclos que se encuentra la señal de salida a nivel alto.

Para la gestión del Monoestable también se declaran las variables de tipo std_logic Disparo, Fin_Cuenta y Est_Anterior. La variable Est_Anterior cuida de que para que el Monoestable actúe, la señal Ping venga del nivel bajo. Cuando esto se produce es la señal Disparo la que desencadena toda la acción. Mientras, Fin_Cuenta desactiva toda esa acción.

Al comenzar la parte descriptiva del proceso se observa un bloque único que solo se ejecuta si se produce un flanco descendente de la señal de reloj Clk mediante una sentencia IF (--Flanco Descendente Reloj). En el interior de este bloque se desarrolla todo el funcionamiento de este circuito.

El primer bloque de condicional que se observa si se cumple la anterior condición discierne si la señal Ping ha tenido una transición de ‘0’ a ‘1’ o si por el contrario, se ha finalizado el tiempo de la señal de salida a nivel alto.

Lo primero se gestiona mediante una sentencia IF que se verifica si se cumple que la variable Est_Anterior es ‘0’ y la señal Ping es ‘1’. Est_Anterior se asigna al final de la ejecución de cada proceso marcando siempre el valor que tenía anteriormente la señal Ping. Por lo tanto, que se cumpla esta condición es un indicador de que se ha producido un flanco ascendente de la señal Ping. En este caso la variable Disparo se pone a nivel alto (--Flanco Ascendente Pulsador).

Lo segundo se gestiona mediante la sentencia ELSIF y se chequea cuando la variable Fin_Cuenta se encuentra a nivel alto. Esta variable se gestiona en el siguiente bloque de comparación y al ponerse a nivel alto indica que se ha finalizado la cuenta de ciclos que se ajusta mediante NUM_CICLOS y que la señal de salida debe de ponerse a nivel bajo. En caso de que esta condición se cumpla la variable Disparo se pondrá a nivel bajo (--Fin Pulso).

En el siguiente bloque de condicional solo se entra si la variable Disparo se encuentra a nivel alto (--Actuación Flanco Asc. Puls.). En su interior se verifica si la variable Cuenta es menor que el dato genérico NUM_CICLOS. Si esta condición es afirmativa, se prosigue la cuenta de ciclos incrementando en uno la variable Cuenta. También se pone la variable Fin_Cuenta a nivel bajo y la señal de salida Pulso se mantiene a nivel alto (--Emisión Pulso durante NUM_CICLOS).

Por el contrario, cuando la variable Cuenta alcanza el valor NUM_CICLOS la variable Cuenta se resetea para la siguiente vez, la variable Fin_Cuenta se pone a nivel alto y la señal Pulso se pone a nivel bajo. Como se ha comentado anteriormente, al ponerse la variable Fin_Cuenta a nivel alto, la variable Disparo se pone a nivel bajo y no se vuelve a ejecutar ninguna acción de cuenta permaneciendo la salida Pulso a nivel bajo (--Finalizar Pulso y Preparar sgte.).

Antes de cerrarse el proceso se realiza la asignación que se adelantó en la que se vuelca el valor de Ping sobre la variable Est_Anterior. Este hecho dota al circuito de cierta memoria para saber cuando debe producirse un disparo real y cuando no. Esta es la principal diferencia entre un circuito físico implementado con un LM555 y este circuito sintetizado en la FPGA.

Finalmente se cierra el proceso y la arquitectura habiendo generado completamente un monoestable que cumple todas las especificaciones

necesarias para excitar, a través de una etapa de potencia al emisor de ultrasonidos.

Cabe mencionar que dado que la frecuencia del oscilador de placa es de 50 MHz., cada ciclo dura 1 / 50.000.000 segundos. Como NUM_CICLOS está fijado a 200.000 ciclos, la señal Pulso permanecerá a nivel alto después de un disparo durante 4 milisegundos.

De tal forma, queda totalmente definida la función e implementación de este último bloque del circuito Medidor descrito que se sintetizará según todo lo mencionado en la FPGA.

CAPÍTULO 4

CONCLUSIONES Y APLICACIONES

4. CONCLUSIONES Y APLICACIONES

En este punto de la memoria se hará un balance del proyecto desarrollado desde distintos puntos de vista; siendo importante valorar la consecución de los objetivos y su verificación de funcionamiento.

Seguidamente se comentarán las diversas aplicaciones que se pueden esperar del Medidor diseñado, así como hacia donde se pueden encauzar futuras ampliaciones para mejorar o complementar el funcionamiento del sistema.

Por último se comparará el funcionamiento y desarrollo de este proyecto mediante una FPGA con lo que se esperaría que hubiera sido con un microprocesador estándar.

4.1. CONCLUSIONES Y RESULTADOS

Retrocediendo a los objetivos que se esperaban del medidor, el fin principal del presente proyecto es la obtención de un sistema capaz de realizar mediciones de distancias y mostrarlas al usuario comprensiblemente.

Para conseguir este objetivo se usaría una FPGA del fabricante Xilinx y modelo XC3S200 montada sobre la placa de desarrollo de la firma Digilent llamada Spartan III; así como elementos para la emisión y recepción de ondas de ultrasonidos.

De tal forma, se ha realizado un desarrollo hardware de una placa auxiliar donde se monten todos los recursos adicionales del medidor, adaptando los elementos y funciones de esta placa auxiliar a la placa Spartan III para poder conseguir un sistema compacto.

También se ha realizado un desarrollo de código en lenguaje VHDL para conseguir que la FPGA mencionada utilice adecuadamente los recursos añadidos en la placa auxiliar y los que existen en la placa Spartan III.

Finalmente se ha integrado todo el sistema para obtener un medidor de distancias por ultrasonidos según las especificaciones dadas.

A lo largo de esta memoria se han ido abordando todos los hitos que suponía el desarrollo del sistema mencionado según especificaciones. Por lo tanto se dan por cubiertos todos los objetivos que se comentan a continuación:

- En primera instancia se precisaba de un sistema hardware que adaptase las señales eléctricas a ultrasonidos y viceversa. Con este fin se ha implementado la placa auxiliar con un sistema astable que oscila a 40 KHz. y con un detector de tonos sintonizado a la misma frecuencia. Con este diseño se pueden enviar y recibir ondas de ultrasonidos mediante emisor y receptor de ultrasonidos. Dicho fin fue cubierto en el Capítulo Segundo y su funcionamiento se verificó sobre laboratorio calibrando los parámetros pertinentes.

- Seguidamente se diseñaría un código en lenguaje VHDL que se cargase sobre la FPGA de la placa de desarrollo Spartan III utilizando sus recursos y los de la placa auxiliar para concluir el objetivo global. Por esto se ha sintetizado un circuito sobre la FPGA capaz de cronometrar tiempos de eco mediante señales externas, de operar sobre ellos y obtener la distancia, de redondearla y, por último, de representarla en el display de la placa. A parte de esto, dicho circuito permite la comunicación con el usuario del medidor para poder actuar sobre su funcionamiento. Este objetivo se desarrolló en el Capítulo Tercero y su funcionamiento se verificó mediante diversas pruebas sobre la placa de desarrollo.

Finalmente, se puede asegurar que todo el diseño realizado a nivel hardware y VHDL funciona adecuadamente. La compatibilidad entre los dos sistemas que fueron probados independientemente, placa Spartan III y FPGA, y placa auxiliar; es total.

Para verificar las especificaciones del medidor se ha llevado a cabo un ensayo de medidas.

De tal modo, se situó el medidor junto a una cinta métrica y mediante una superficie plana que se interponía en el haz de ultrasonidos del medidor, se fueron realizando distintas medidas comprobando precisión, repetitividad, histéresis y rango máximo.

Finalmente se obtuvo la tabla de resultados que se expone a continuación:

Medida Real	Lecturas Medidor		
	1	2	3
5 cm.	6 cm.	7 cm.	9 cm.
6 cm.	8 cm.	9 cm.	8 cm.
7 cm.	9 cm.	9 cm.	8 cm.
8 cm.	10 cm.	11 cm.	11 cm.
9 cm.	12 cm.	12 cm.	13 cm.
10 cm.	13 cm.	14 cm.	13 cm.
13 cm.	14 cm.	15 cm.	14 cm.
16 cm.	17 cm.	18 cm.	18 cm.
19 cm.	20 cm.	20 cm.	20 cm.
21 cm.	22 cm.	21 cm.	22 cm.
24 cm.	26 cm.	26 cm.	26 cm.
27 cm.	28 cm.	27 cm.	28 cm.
30 cm.	30 cm.	31 cm.	31 cm.
35 cm.	36 cm.	36 cm.	36 cm.
40 cm.	42 cm.	41 cm.	41 cm.
45 cm.	47 cm.	46 cm.	47 cm.
50 cm.	51 cm.	52 cm.	52 cm.
55 cm.	56 cm.	56 cm.	56 cm.
60 cm.	61 cm.	62 cm.	62 cm.
65 cm.	66 cm.	66 cm.	66 cm.
70 cm.	71 cm.	71 cm.	72 cm.
75 cm.	75 cm.	75 cm.	75 cm.
80 cm.	81 cm.	81 cm.	82 cm.
85 cm.	85 cm.	85 cm.	86 cm.
90 cm.	90 cm.	91 cm.	92 cm.
95 cm.	96 cm.	96 cm.	96 cm.
1 m.	1 m.	1 m.	1 m.
1,1 m.	1,1 m.	1,1 m.	1,1 m.
1,2 m.	1,2 m.	1,2 m.	1,2 m.
1,3 m.	1,3 m.	1,3 m.	1,3 m.
1,4 m.	1,4 m.	1,4 m.	1,4 m.
1,5 m.	1,5 m.	1,5 m.	1,5 m.
1,6 m.	1,6 m.	1,6 m.	1,6 m.
1,7 m.	1,7 m.	1,7 m.	1,7 m.
1,8 m.	1,8 m.	1,8 m.	1,8 m.
1,9 m.	1,9 m.	2 m.	1,9 m.
2,0 m.	2 m.	2 m.	2,1 m.
2,1 m.	2,1 m.	2,1 m.	2,1 m.
2,2 m.	2,2 m.	2,2 m.	2,2 m.
2,3 m.	2,3 m.	2,3 m.	2,3 m.
2,4 m.	2,4 m.	2,4 m.	2,5 m.
2,5 m.	2,5 m.	2,5 m.	2,5 m.

Tabla 4.1: Resultado del ensayo del medidor

En esta tabla se observa como para cada medida real de la cinta métrica, se toman tres medidas del sistema medidor. Las dos primeras medidas se toman cuando la distancia se va haciendo aumentar, mientras que la tercera media se toma cuando la distancia se va disminuyendo.

En cuanto a las características del ensayo, también se puede apreciar que se van modificando los incrementos en la distancia según diversos intervalos.

Los resultados que se deducen de esta tabla son los siguientes:

- *Precisión:* El medidor posee un error aproximado de dos centímetros para las medidas en rango de centímetros y prácticamente nulo en el de metros. Este error siempre se produce aumentando la medida, nunca disminuyéndola del valor real. También se observa que hasta rebasar los 15 centímetros el error es mayor.

- *Repetitividad:* En la tabla se puede observar que la repetitivad es muy grande, tanto que en ocasiones se realizan las mismas medidas en las tres lecturas.

- *Histéresis:* Como ya se ha comentado, las tres lecturas de cada medida real en la tabla son repetitivas. Es decir, que la tercera no tiende a variar de las dos primeras, concluyendo en que el medidor no posee histéresis.

- *Rango máximo:* Según el gran error que se observa en las medidas por debajo de 15 centímetros, se fijará este dato como la mínima lectura fiable. Por otra parte, el mayor dato registrado con cierta estabilidad ha sido el de 2,5 cm. fijándose así este dato como mayor distancia fiable.

4.2. APLICACIONES COMERCIALES Y EXPERIMENTALES

El sistema de medida implementado cubre ciertos requerimientos comerciales por sí solo. Este Medidor de Distancia puede realizar mediciones en rangos medios a velocidad elevada y con una precisión media.

Concretamente, con un medidor de distancias con unas características o especificaciones como las que reúne este proyecto se podrán cumplir eficazmente tareas como las siguientes:

- Tasar metros cuadrados de una vivienda.

- Medir distancia entre objetos difícilmente accesibles (techos, etc.).

- Medir objetos con una longitud media rápidamente.

- Obtener distancia de un objeto móvil al medidor (garajes, puertas, etc.).

Existen multitud de sistemas de medición de distancias implementados con diversas técnicas en el mercado. Lo más destacable del medidor diseñado es la capacidad para la medición continua y la explotación de los recursos de la placa Spartan III, a parte del reducido tiempo de desarrollo.

Sin embargo, elaborando una serie de medidores con placas específicamente para fines comerciales se podría desarrollar un sistema de medición mucho más competitivo, incluso cubriendo mayores rangos de medida. De este modo, se reduciría el tamaño y se dotaría al sistema de baterías y un display que permitiese más funciones y, en definitiva, se desarrollaría un sistema más compacto.

Por otra parte, no son menos interesantes las aplicaciones experimentales que este sistema puede tener en el campo del desarrollo de sistemas automáticos más complejos integrándose en ellos. Esto es posible debido a que el potencial de la FPGA permite seguir sintetizando sistemas en su interior; puesto que su porcentaje actual de utilización es del 48 %.

De tal modo podría servir, por ejemplo, para situar piezas en una cadena de montaje antes de realizar una intervención sobre ella. En un vehículo, teniendo en cuenta la velocidad del mismo y con un mayor rango de medida, podría indicar cual es la distancia a obstáculos en la trayectoria descrita e incluso ajustar la velocidad.

Llegando a ese punto, el sistema se puede integrar en un amplio abanico de robots en fase experimental, como pequeños sistemas que salvan obstáculos donde la misma FPGA gestionase también los movimientos del robot.

Pero también, incluyendo en el sistema servos que muevan la placa donde se encuentra el par emisor-receptor y la lógica para controlarlos e interpretar los resultados; se puede llegar a obtener cierto grado de visión por ultrasonidos.

Para ello se determinarán ciertas posiciones de medición según eje X e Y. Interpolando en una tabla todos los datos de distancias muestreados, se obtendrá una matriz de puntos tridimensional más exacta aunque más lenta cuanto más mediciones se realicen. Cada medición y tratamiento de la misma llevaría aproximadamente 18 milisegundos.

Todo este procesamiento podría ser realizado en la FPGA pasando por un canal de comunicaciones a un procesador central para su análisis el resultado de la matriz de “visión”.

En cualquier caso, sin olvidar una de las mayores ventajas del lenguaje de descripción hardware VHDL, la reutilización del código generado en este proyecto puede intervenir en el desarrollo de otras aplicaciones de muy distantes a la actual.

Con pocas variaciones se tendrá un controlador del display para representar números o caracteres cualesquiera en sus cuatro dígitos. Igualmente se dispone de un registro con clear, un multiplicador o un redondeador. Es interesante el contador con control de cuenta y parada, clear e indicador de desborde. Por último, el modelo del autómata también puede ser reutilizado fácilmente.

4.3. AMPLIACIONES FUTURAS

Habiendo cumplido todos los objetivos y especificaciones que este proyecto exigía; se ha obtenido un sistema abierto y sensible a diversas mejoras que, por motivo de tiempo y requerimientos, no se han llevado a cabo.

Algunas de estas mejoras ampliarían el rango de aplicaciones de este sistema de medición por ultrasonidos, como se ha comentado en el anterior epígrafe. Otras mejoras sobre el sistema básico aportarían a este más funciones y fiabilidad, e incluso ser un producto más económico al llevarlo al ámbito comercial.

Todas estas posibles mejoras y ampliaciones se enumeran y comentan una por una a continuación:

- *Calibración manual:* Este código VHDL fue desarrollado durante el diseño del medidor pero no se logró hacer funcionar y por tiempo se descartó. Dicho sistema consta de algunos estados más en el Secuenciador, ciertas señales de control adicionales, un Procesador más complejo (dos registros más, un divisor y un multiplexor) y un Conversor con algo más de código. Externamente, mediante un interruptor en la placa, se selecciona el modo de calibración una vez el usuario se haya cerciorado de colocar el medidor a un metro de distancia de un obstáculo. Entonces se emite una onda de ultrasonidos. Con la distancia conocida y sabiendo el tiempo de eco se halla la velocidad del sonido y se muestra en el display hasta que el interruptor se devuelva al modo de medición.

- *Autocalibración:* La realización de la autocalibración supone que el medidor sepa en cada momento el valor de la velocidad del sonido

exacto para calcular la distancia. La implementación de dicho sistema necesita de un sistema secundario de adquisición de información a través de sensores de temperatura, entre otros. Conociendo el valor de la temperatura se puede saber como varía la velocidad del sonido en una atmósfera estándar, suponiendo estable su composición, etc. Este sistema aumentaría la precisión de la medida notablemente siempre y cuando estuviese muy perfeccionado.

- *Reducción de placa auxiliar*: Esta ampliación en código VHDL conlleva a la reducción de complejidad y circuitos analógicos en la placa auxiliar de este proyecto. De tal forma, se eliminarán circuitos integrados como el LM555, LM567 o LM311. Por lo tanto, en el interior de la FPGA se generará la frecuencia de resonancia de 40 KHz. sustituyendo al LM555. También en la FPGA se sintetizará un circuito que por sincronismos compare la onda recibida con una de 40 KHz. y discierna si se ha recibido el eco o no evitando falsos disparos sustituyendo a los LM567 y LM311. En la placa auxiliar solamente quedaría una etapa de potencia con un transistor para excitar al emisor y un amplificador para ampliar la señal recibida. Esta mejora haría mucho mas fiable y potente el sistema ya que evitaría su descalibración de frecuencias permanente o por temperatura. Su serie comercial también sería mucho más compacta y económica al ahorrar componentes y pistas en placa.

- *Ampliación del rango de medida:* El rango de medida de este Medidor de Distancia está limitado por el par receptor-emisor de ultrasonidos. La FPGA funcionaría con cualquier rango cambiando parámetros en su código. De tal modo, si se introduce otro par receptor-emisor se podrá ampliar este rango de medición. También se puede adaptar el sistema a otro tipo de medición como la radiofrecuencia. Además se tendrá que variar la especificación de muestreo en el display y las dimensiones de los vectores de los componentes, modificando el código VHDL.

- *Transmisión de datos:* Este desarrollo supone el dotar a la FPGA de una UART para transmisión de datos. Esta transmisión se realizaría con un protocolo RS232 usando los conversores de la placa de desarrollo, o mediante un protocolo USB o sistema de bus (Profibus, CANBus, etc.) adaptando niveles con circuitos externos. Los datos se transmitirían con un formato distinto al que se representan, es decir, se enviarían en código ASCII con las letras cm. o M. después de los números. Por esta UART también se podrían recibir datos como la calibración o para que la FPGA ejecute cualquier tipo de acción como

realizar una medición o no. De tal forma se habrá desarrollado un sistema adquisición de datos para conectar a un sistema como un PC, autómata, etc.

4.4. COMPARATIVA DE DESARROLLO CON MICROPROCESADOR

A lo largo de esta memoria se ha comentado todo el desarrollo del Medidor de Distancia según especificaciones y siempre teniendo en cuenta su implementación en una FPGA y la codificación en VHDL.

Sin embargo, este mismo proyecto podría haber sido diseñado mediante un microprocesador que es la otra opción alternativa a la FPGA. En el mercado, cada día se imponen más los sistemas basados en FPGA, pero los microprocesador siguen siendo sistemas competitivos.

A continuación se comentan ventajas e inconvenientes de cada sistema comparándolos.

Lo primero a destacar es que, como norma general un sistema en VHDL sobre FPGA suele conllevar un mayor tiempo de desarrollo. Esto se debe a que en estos sistemas se deben generar las bases y filosofía hardware a seguir en el diseño.

Sin embargo, un microprocesador se puede programar con lenguajes de alto nivel con funciones muy definidas y que aportan muchas facilidades. Si se busca un time to market reducido lo mejor es decantarse por un microprocesador.

Por otra parte, la FPGA es un sistema muy abierto en el que se puede generar cualquier circuito que se necesite hasta agotar los recursos, lo cual puede suplirse con una FPGA superior.

Esto conlleva a que en este proyecto no se cuente con una FPGA pero que en cuanto se desee puede incluirse en la misma, así como diversos sistemas de calibración, etc. Y todo ello sin cambiar de hardware.

Pero esto tiene un inconveniente principal que es la propagación y acotación de retardos. Siempre que se realiza un diseño en una FPGA hay que contar con ellos y generar sistemas de sincronismo para paliar sus efectos. Un sistema bien diseñado puede no funcionar por este motivo.

En un microprocesador todo el hardware es cerrado y definido según especificaciones del mismo. Por lo tanto nunca se podrán ampliar los recursos

y cambiar un diseño para ampliar alguna funcionalidad del mismo supone el cambio de microprocesador con todas las repercusiones que supone.

Esta desventaja acompaña a una ventaja fundamental en los microprocesadores, esta es que cuando se adquiere uno se puede utilizar directamente sin preocuparse de otra cosa que no sea el software que se le va a cargar. No es necesario pensar en retardos del hardware ni ningún otro factor de este tipo, simplemente programarlo.

Lo anteriormente comentado, lleva a que una FPGA contiene un circuito totalmente opaco y desconocido al exterior, mientras que un microprocesador queda totalmente definido con solo mirarle el modelo en su encapsulado. Cuando en un sector es muy importante la innovación y no revelar información sobre un desarrollo que pueda ser plagiada, la mejor opción es una FPGA.

Igualmente, la velocidad de funcionamiento máxima de una FPGA depende siempre de los circuitos sintetizados en su interior y de la eficiencia de los mismos. De tal forma, una FPGA puede funcionar a velocidades muy elevadas. En el caso de los microprocesadores, la frecuencia de funcionamiento viene definida en las especificaciones y en ocasiones llega a ser relativamente lento. Sin embargo, esto siempre dependerá del circuito a sintetizar en FPGA y del software a programar en microprocesador.

Para el desarrollo del código VHDL o del programa del microprocesador, su síntesis, compilación y programación, se cuenta con una herramienta de desarrollo. En esta memoria se comentó el funcionamiento básico de la herramienta adecuada para la FPGA utilizada. Comparando estas herramientas con las de microprocesadores; las de la FPGA son mucho más avanzadas, aportan muchos mas sumarios e informes, poseen posibilidades de soporte on-line y permiten una simulación extremadamente realista.

La programación de la FPGA posee un punto débil, este es la necesidad de añadir hardware adicional para la programación no volátil. Un microprocesador no necesita como norma general ningún dispositivo adicional para ser programado. A este respecto, si supone un incremento económico elevado o complejidad no deseada, la mejor opción es un microprocesador.

Por otra parte, cabe mencionar los avances que se han realizado en el desarrollo de códigos para FPGA. En este proyecto no se ha hecho uso de ellos, pero existen componentes a bajo coste o libres para poder ser copiados directamente en el código VHDL y utilizarlos en base a sus hojas de características como si fuera un componente real.

También se han desarrollado aplicaciones para la generación y carga de los llamados COREs en la FPGA. De esta forma se puede conseguir introducir elementos como un microprocesador comercial en el interior de la FPGA, programarlo y además añadir en la misma FPGA recursos adicionales (en VHDL u otros COREs). Así se obtendrá un sistema extremadamente compacto y opaco al exterior combinando las ventajas de la FPGA y los microprocesadores en un mismo circuito.

Sin duda, hoy día para un desarrollo ampliable, abierto, complejo, compacto, seguro y fiable; la mejor opción es la FPGA. Siempre se debe tener en cuenta que, el tiempo de desarrollo será mayor, pero los beneficios obtenidos a cambio merecerán la pena.

ANEXO A: MODO DE USO Y FUNCIONAMIENTO

En el presente anexo se comentarán las funciones que posee el medidor comentando como actuar sobre ellas y que tipo de resultado se obtiene.

Por lo tanto, este anexo será una síntesis de todo lo expuesto en clave técnica anteriormente, tratando de simplificar al máximo y desde la perspectiva del usuario final.

Las cuatro funciones básicas que de cara al usuario realiza este medidor son: encendido / apagado, selección de modos de lectura, reseteo y autoescalado. A continuación se comentarán cada una de estas funciones brevemente.

Identificación de Elementos

Para poder actuar sobre el Medidor como se describirá en este capítulo se han asignado a los recursos de la placa Spartan III determinadas funciones.

Entre estos recursos se encuentran los de actuación que son pulsadores e interruptores y los indicadores que son LEDs y el display de 4 dígitos. Con la asignación de ciertas propiedades en función del estado del Medidor a estos recursos se tendrá el control total del Medidor de Distancia.

De los efectos que tiene la acción de cada dispositivo de entrada o el significado de cada dispositivo de salida se hablará adelante. No obstante, a continuación se muestra la figura A.1, donde se localizan todos los recursos de la placa Spartan III asignados al Medidor:

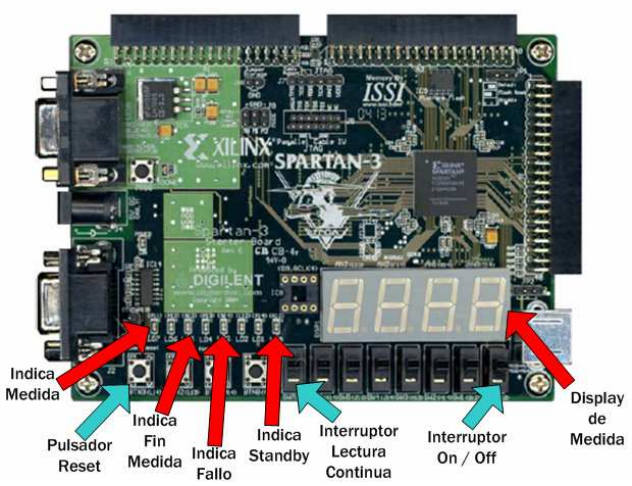


Figura A.1: Elementos del Medidor de Distancia

Como se observa en esta figura, existen flechas rojas y azules. Las flechas azules engloban a los pulsadores e interruptores, es decir, a los elementos de control del sistema. Las flechas rojas indican los nombres de todos los dispositivos indicadores de estado y de medida, es decir, las salidas del sistema hacia el usuario.

Con esta figura quedan identificados todos los elementos de la placa Spartan III necesarios a la hora de usar el Medidor de Distancia. En adelante se hará referencias a estos elementos para explicar el funcionamiento global del medidor.

Encendido / Apagado

Esta función permite la habilitación del Medidor o desconexión dependiendo de si este se está usando o no.

Cuando el medidor está encendido y no se realizan medidas se muestra la última medición realizada en el Display de Medida. Si no se ha realizado ninguna medida desde que se encendió o reseteó por última vez el medidor, dicho display marcará 0 centímetros. Además también permanecerá encendido el LED indicador de Fin de Medida.

Si se desea ahorrar este consumo y actividad innecesarios, con esta función se podrá apagar el Medidor cuando se desee; permaneciendo entonces en un estado de Standby donde el consumo será mínimo. Actualmente el Medidor se conecta a la red, pero si en algún momento se alimenta mediante baterías, esta función será muy útil para la autonomía del medidor.

Por lo tanto, para encender el Medidor de Distancia se debe desplazar hacia el Display de Medida el Interruptor On / Off que se muestra en la figura 4.1. En ese momento el Medidor quedará en el estado dispuesto anteriormente a espera de comenzar a realizar mediciones.

Si el Interruptor On / Off se desplaza hacia el borde de la placa, a su posición de inactividad, el Medidor se apagará. En este momento el Display de Medida quedará en blanco y se borrará cualquier medida que haya en él sin poderse recuperar. Todos los indicadores permanecerán apagados menos el Indicador Standby. Ahora el Medidor queda en espera de ser encendido cuando el usuario lo desee.

Modos de Lectura

Una de las mejores características de este medidor es su capacidad de realizar mediciones continuas, es decir, una tras otra sin tener realizar ninguna acción sobre él.

Sin embargo, también puede cortarse dicho flujo continuo para realizar tantas mediciones como se desee y pararlo en cualquier momento.

Por lo tanto, existe un modo de lectura continuo en el que cada 340 milisegundos aproximadamente se realiza una medición cuyo resultado se muestra inmediatamente en el Display de Medida. Entre una medida y otra se encienden el Indicador Medida mientras la onda de sonido viaja y el Indicador Fin Medida cuando se recibe la onda y se espera cierto tiempo a volver a realizar una medida.

Si no se desea un modo de lectura continuo, en la última medición realizada el Indicador Fin Medida queda encendido indefinidamente hasta que se vuelva a lanzar otra medida. En ese momento se enciende el Indicador Medida.

Estos son los dos modos de lectura que existen pero, para seleccionar uno u otro habrá que actuar sobre un interruptor en la placa Spartan III. Este es el Interruptor Lectura Continua que se encuentra a la izquierda del bloque de interruptores.

En el momento que este interruptor se desplace hacia el Display de Medida, quedará seleccionado el modo de lectura continuo y mientras tanto se realizarán medidas cada 340 milisegundos.

Pero si este interruptor se desplaza en sentido contrario al descrito, el Medidor detendrá su ejecución quedando enclavada en el Display de Medida la última medida realizada indefinidamente. Esto evita tener que anotar la medida. Además también se puede realizar una sola medida al actuar sobre el interruptor poniéndolo en lectura continua e inmediatamente después devolviéndolo a su posición inicial.

Reseteo

Esta es una función que, por norma general, no se tiene por que usar nunca. Sin embargo, es conveniente que todo sistema electrónico posea un mecanismo de reseteo para poder recobrar el control si este se pierde por cualquier circunstancia o si se quiere reiniciar el sistema directamente.

Por lo tanto, no siendo menos, este Medidor de Distancia posee un pulsador que detiene cualquier ejecución que se de o vaya a dar en el sistema. Cuando se actúa sobre este pulsador se borra cualquier medida que haya en el Display de Medida perdiéndose. Así, dicho display queda en blanco y el único LED que permanece encendido es el Indicador Fin Medida.

Una vez se deja en reposo este pulsador se vuelve al mismo estado que cuando se enciende el sistema. En ese momento, se actuará en consecuencia de cómo se encuentre el Interruptor Lectura Continua realizando lo oportuno.

El Pulsador Reset se encuentra en la parte inferior izquierda de la placa Spartan III e incluso, su posición está indicada en la serigrafía de la placa para no confundirlo. Con una leve pulsación sobre este dispositivo el Medidor de Distancia será reiniciado perdiendo cualquier medida realizada.

Autoescalado

Esta función se realiza de forma automática. No es necesario actuar sobre ningún dispositivo de la placa para llevar a cabo esta función. Cada vez que se realiza una medida, la escala del resultado será seleccionado de forma autónoma y representado en el Display de Medida coherentemente con esta selección.

Aunque este recurso es automático, sí que conviene comentar cual es su resultado y funcionamiento. Será también importante diferenciar cuando se está representando en una escala o en otra.

De tal forma, una vez que se ha realizado una medición y se va a representar, si el valor medido se encuentra entre 5 y 99 centímetros; la escala en la que se muestra en el Display de Medida será la de centímetros.

Cuando una medida se representa en la escala de centímetros, en el Display de Medida se muestra un símbolo semejante a cm. y un punto tras el valor numérico de la medida. Este formato se observa en la figura A.2 que se muestra a continuación:

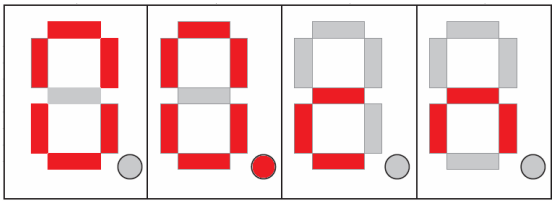


Figura A.2: Símbolo de centímetros

Si un valor medido se encuentra por encima de los 99 centímetros, pero se encuentra también por debajo de 6 metros, la escala en la que se mostrarán los datos en el Display de media será la de metros.

Para representar una medida en escala de metros, se muestran dos símbolos complementarios asemejándose a una M. en el Dislpay de Medida. El valor numérico de la medida consta de un número entero que indica metros y un decimal que indica decímetros. Esta representación se muestra en siguiente figura A.3:

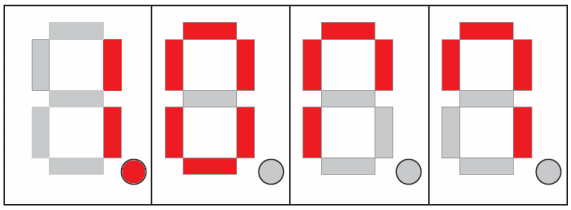


Figura A.3: Símbolo de metros

Siempre que el valor de una medida no se encuentre en el rango de medición especificado para este Medidor, es decir, desde los 5 centímetros a los 6 metros; el sistema lo tomará como un error de medición. Cuando se produzca un error de este tipo, se mostrará un mensaje de error en el Display de Medida con el mismo formato que la figura A.4 que se muestra a continuación:

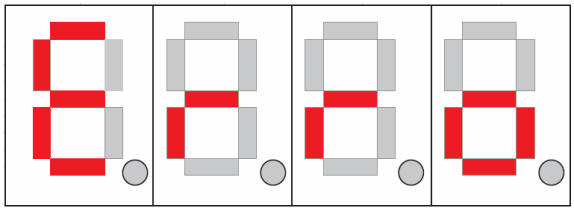


Figura A.4: Mensaje de error

ANEXO B: DOCUMENTACIÓN DEL DVD

En el DVD que acompaña a esta memoria se podrá encontrar todo el software necesario para poder corroborar el funcionamiento y desarrollo del proyecto.

En detalle, los elementos que contiene este DVD en función de sus carpetas son los siguientes:

- *Capture:* Archivos de proyecto y esquemáticos asociados de la Placa Auxiliar. El principal archivo a abrir es *Placa Auxiliar.opj*.

- *Layout:* Archivos para el layout de la Placa Auxiliar. El diseño final de dicha placa se encuentra en *Placa Auxiliar-1.max*. La librería de componentes desarrollada es *Proyecto.llb*.

- *VHDL:* Código completo de VHDL desarrollado tal y como se comentó en la memoria. Todos los nombres de los archivos *.vhd* coinciden con los de la memoria.

- *Programación:* Archivo de programación volátil *Medidor.bit* de la FPGA y no volátil *Medidor.mcs* de la memoria Flash. También se adjunta el archivo *Medidor.msk* necesario para programar.

- *Placa Spartan III:* Hoja de características y manual de usuario de la placa de desarrollo utilizada para la implementación del proyecto. En formato *.pdf*.

- *Datasheets:* Hojas de características y manual de usuario de todos los componentes utilizados en este proyecto. En formato *.pdf*.

- *Memoria:* Esta misma memoria en el archivo *Memoria PFC.pdf*.

- *ISE WebPACK 8.1i:* Instalación del software de desarrollo de libre distribución para la FPGA con el que se ha trabajado y Service Pack adjunto.

BIBLIOGRAFÍA:

Artículos:

[1] Borujeni, S.E. Ultrasonic underwater depth measurement. Isfahan, Iran. Underwater Technology, 2002. Procedings of the 2002 International Symposium on. ISBN: 0-7803-7397-9.

[2] Malaoui, A. Quotb, K. Auhmani, K. Ankrim, M. Benhayoun, M. An accurate electronic device for ultrasonic measurements using a microcontroller. Marseille, France. Industrial Technology, 2004. IEEE ICIT '04. 2004 IEEE International Conference on. ISBN: 0-7803-8662-0.

Libros:

[3] *HDL chip design: a practical guide for designing, synthesizing and simulating ASICs and FPGAs using VHDL or Verilog*. Smith, Douglas J. Ed. Doon, 1997.

[4] *A designer's guide to VHDL synthesis*. Ott, Douglas E. Ed. Kluwer Academi, 1997.

[5] *VHDL modeling for digital design synthesis*. Hsu, Yu-Chin. Ed. Kluwer Academi, 1995.

[6] *VHDL coding styles ans methodologies*. Cohen, Ben. Ed. Kluwer Academi, 1995.

[7] *Printed circuit board design techniques for EMC compliance: a handbook for designers*. Montrose, Mark I. Ed. IEEE Pres, 2000.

Proyectos Fin de Carrera:

[8] Esther San Andrés Jaime. Tutor: Enrique San Millán Heredia. Diseño en VHDL del Core del Microcontrolador 8051.

[9] Antonio José Novillo López. Tutor: José María Armingol Moreno. Localización de Robots móviles Mediante Sensores de Ultrasonidos. PFC Junio 1998.

Recursos de Internet:

[10] http://www.Xilinx.com/ Página oficial de Xilinx.